本节内容

希尔排序

希尔排序（Shell Sort）

最好情况：原本就有序

	10	20	30	40	50	60	70	80
0	1	2	3	4	5	6	7	8

比较好的情况：基本有序

	20	30	10	50	60	40	70	80
0	1	2	3	4	5	6	7	8

希尔排序：先追求表中元素部分有序，再逐渐逼近全局有序

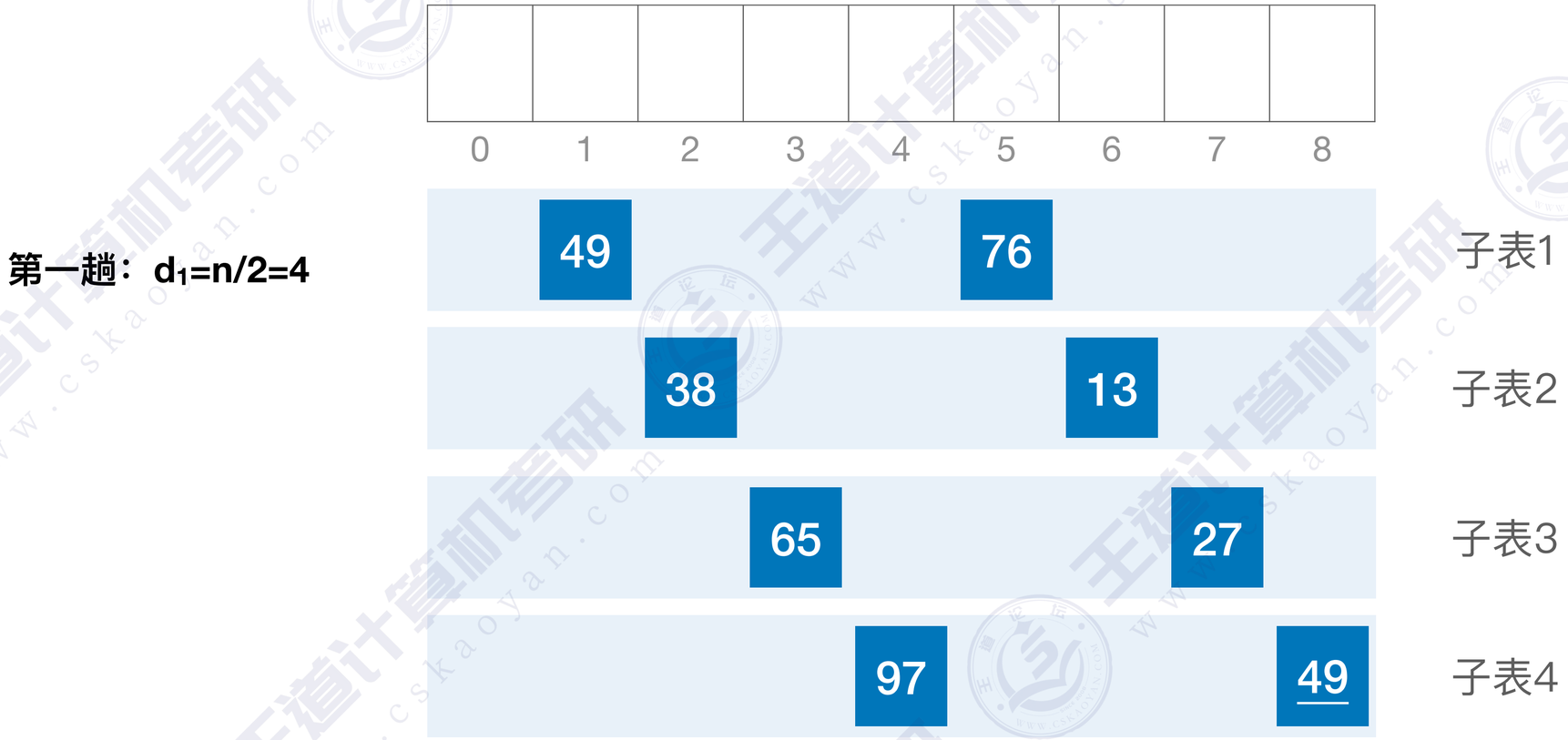
希尔排序（Shell Sort）

希尔排序：先将待排序表分割成若干形如 $L[i, i+d, i+2d, \ldots, i+kd]$ 的“特殊”子表，对各个子表分别进行直接插入排序。缩小增量d，重复上述过程，直到d=1为止。

第一趟：$d_1=n/2=4$

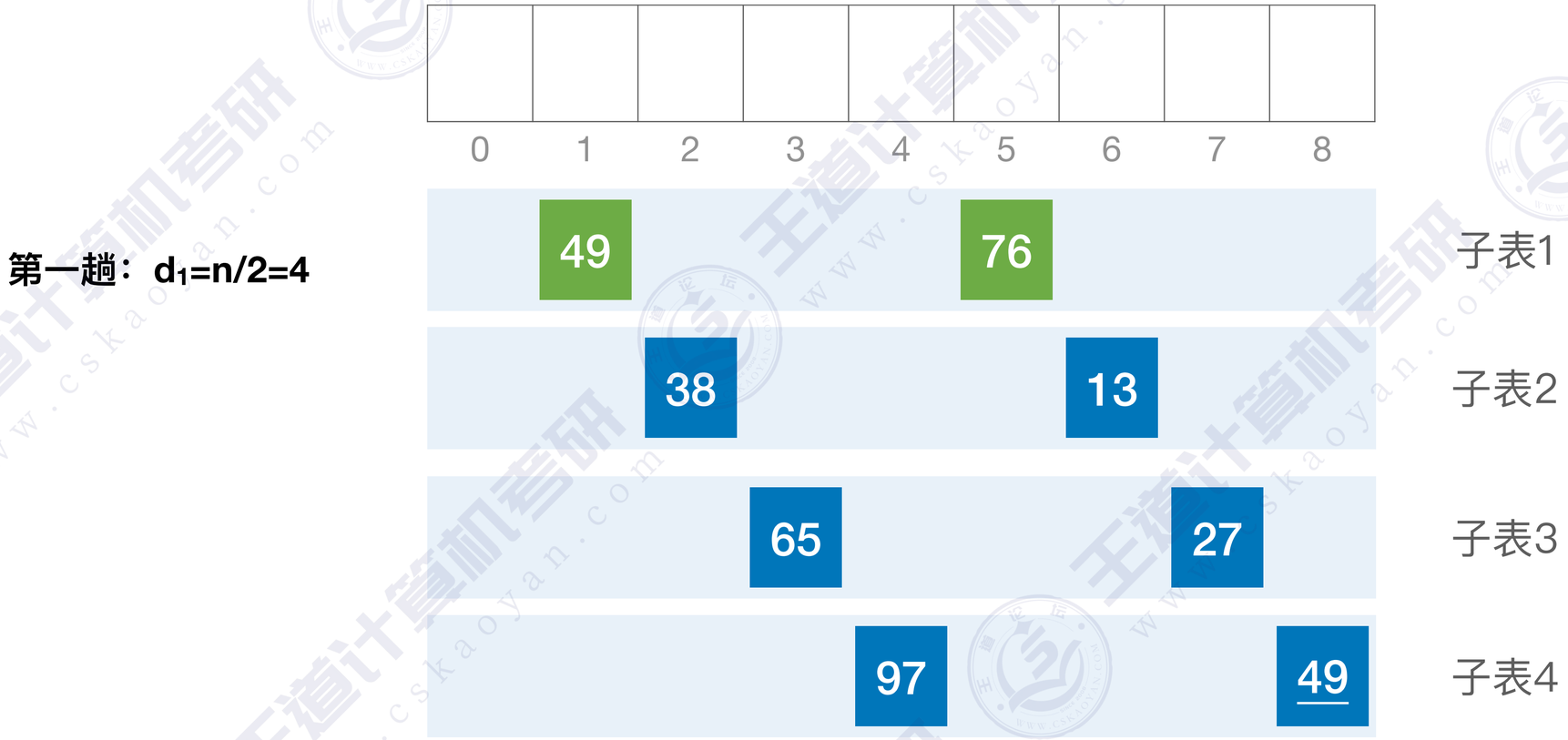
希尔排序（Shell Sort）

希尔排序：先将待排序表分割成若干形如 $L[i, i+d, i+2d, \ldots, i+kd]$ 的“特殊”子表，对各个子表分别进行直接插入排序。缩小增量d，重复上述过程，直到d=1为止。



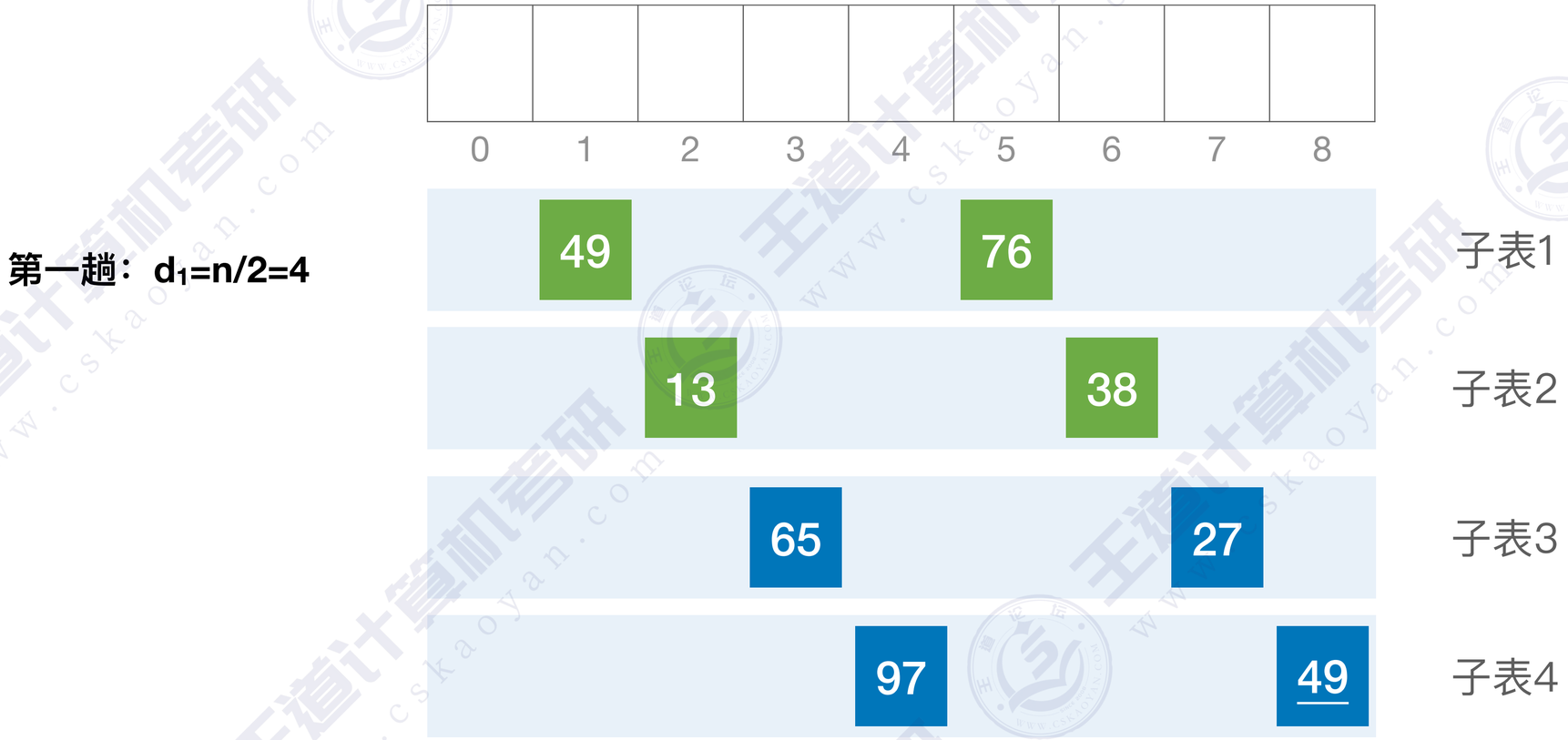
希尔排序（Shell Sort）

希尔排序：先将待排序表分割成若干形如 $L[i, i+d, i+2d, \ldots, i+kd]$ 的“特殊”子表，对各个子表分别进行直接插入排序。缩小增量d，重复上述过程，直到d=1为止。



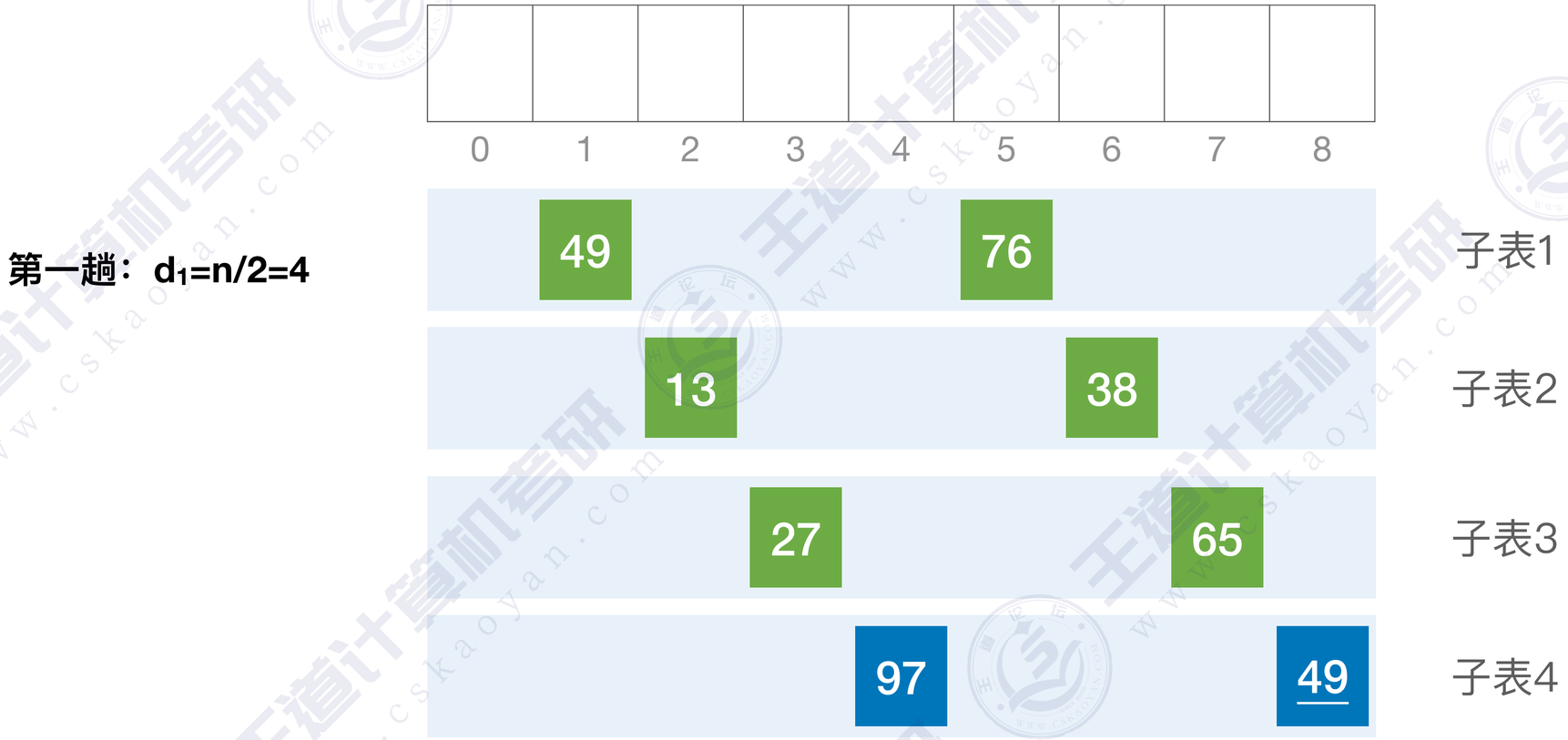
希尔排序（Shell Sort）

希尔排序：先将待排序表分割成若干形如 $L[i, i+d, i+2d, \ldots, i+kd]$ 的“特殊”子表，对各个子表分别进行直接插入排序。缩小增量d，重复上述过程，直到d=1为止。



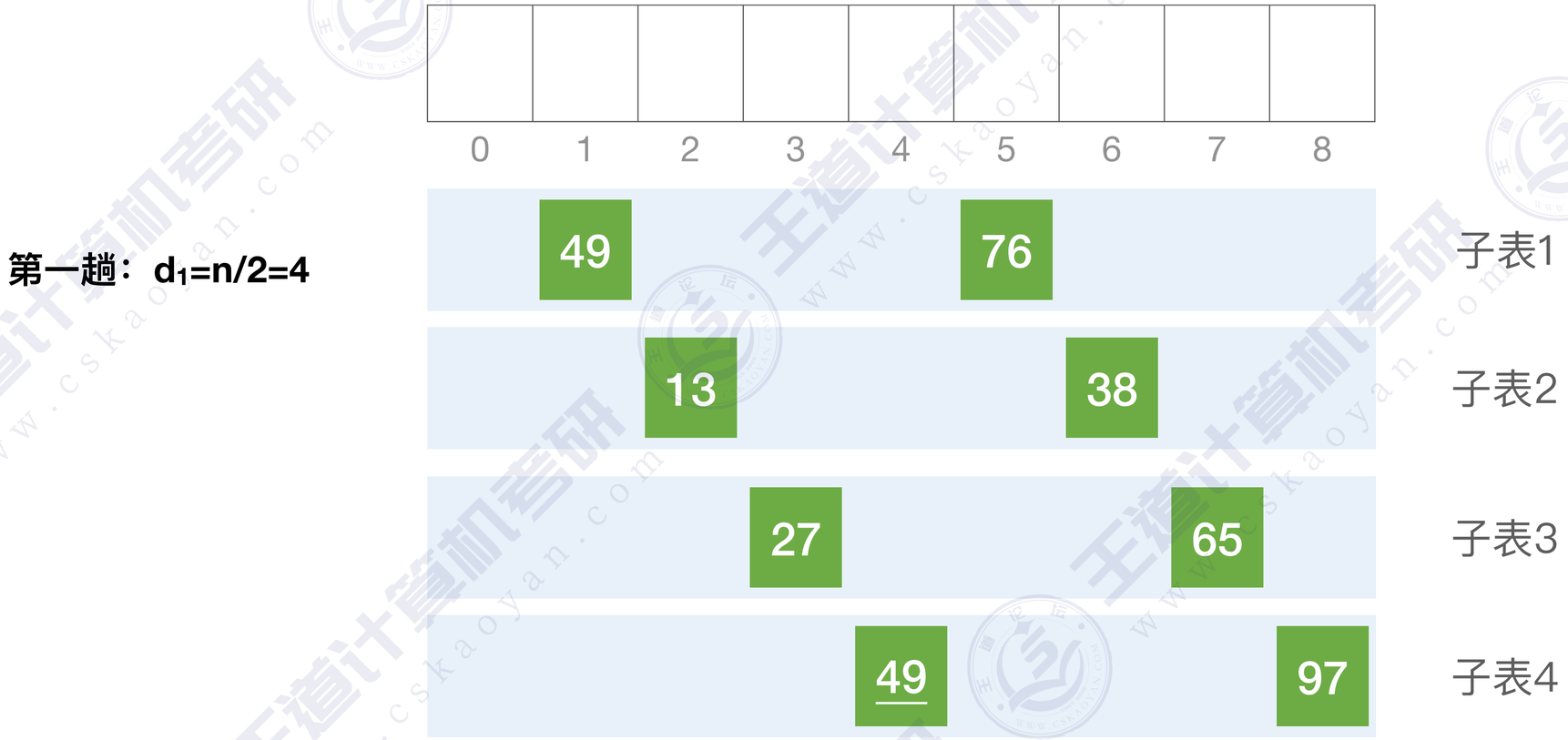
希尔排序（Shell Sort）

希尔排序：先将待排序表分割成若干形如 $L[i, i+d, i+2d, \ldots, i+kd]$ 的“特殊”子表，对各个子表分别进行直接插入排序。缩小增量d，重复上述过程，直到d=1为止。



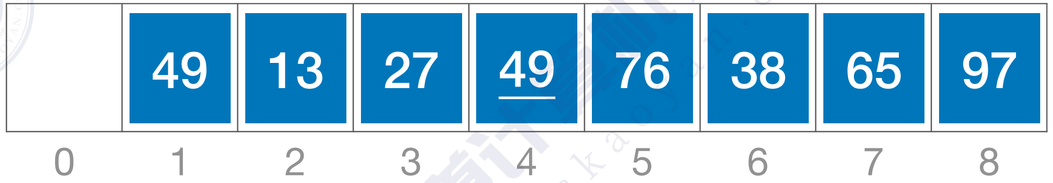
希尔排序（Shell Sort）

希尔排序：先将待排序表分割成若干形如 $L[i, i+d, i+2d, \ldots, i+kd]$ 的“特殊”子表，对各个子表分别进行直接插入排序。缩小增量d，重复上述过程，直到d=1为止。



希尔排序（Shell Sort）

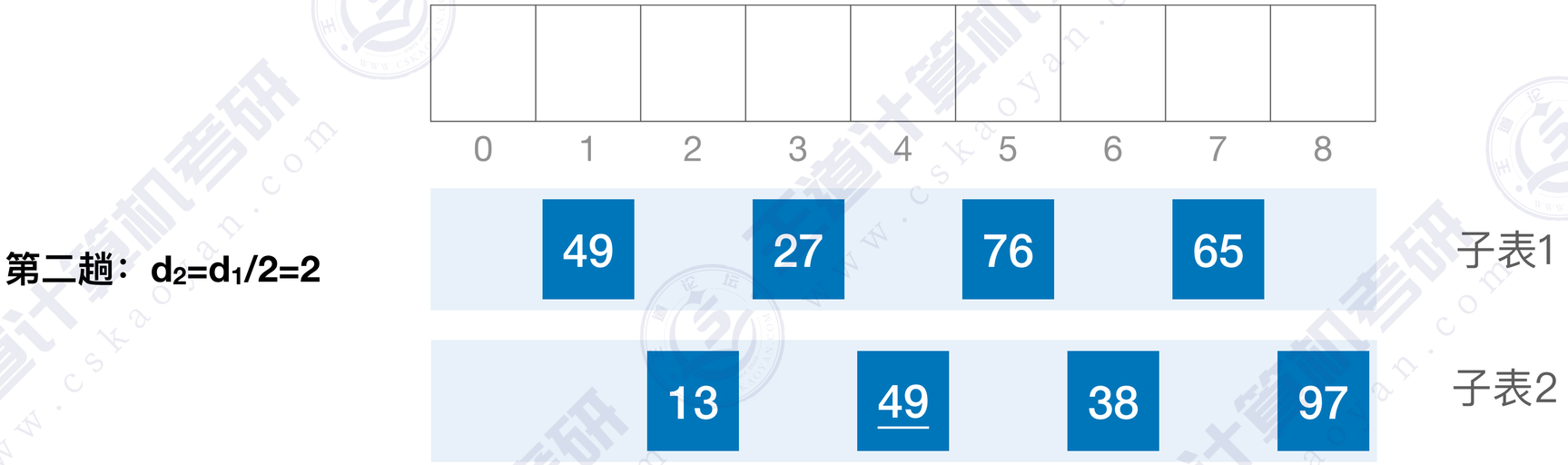
希尔排序：先将待排序表分割成若干形如 $L[i, i+d, i+2d, \ldots, i+kd]$ 的“特殊”子表，对各个子表分别进行直接插入排序。缩小增量d，重复上述过程，直到d=1为止。



第二趟：$d_2=d_1/2=2$

希尔排序（Shell Sort）

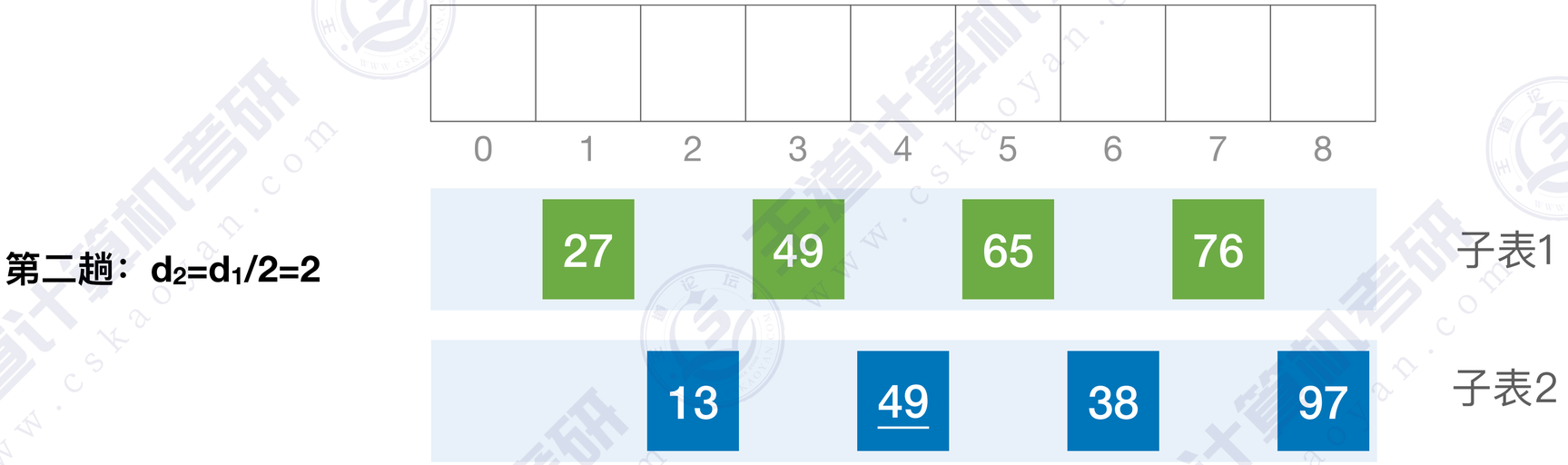
希尔排序：先将待排序表分割成若干形如 $L[i, i+d, i+2d, \ldots, i+kd]$ 的“特殊”子表，对各个子表分别进行直接插入排序。缩小增量d，重复上述过程，直到d=1为止。



第二趟：$d_2=d_1/2=2$

希尔排序（Shell Sort）

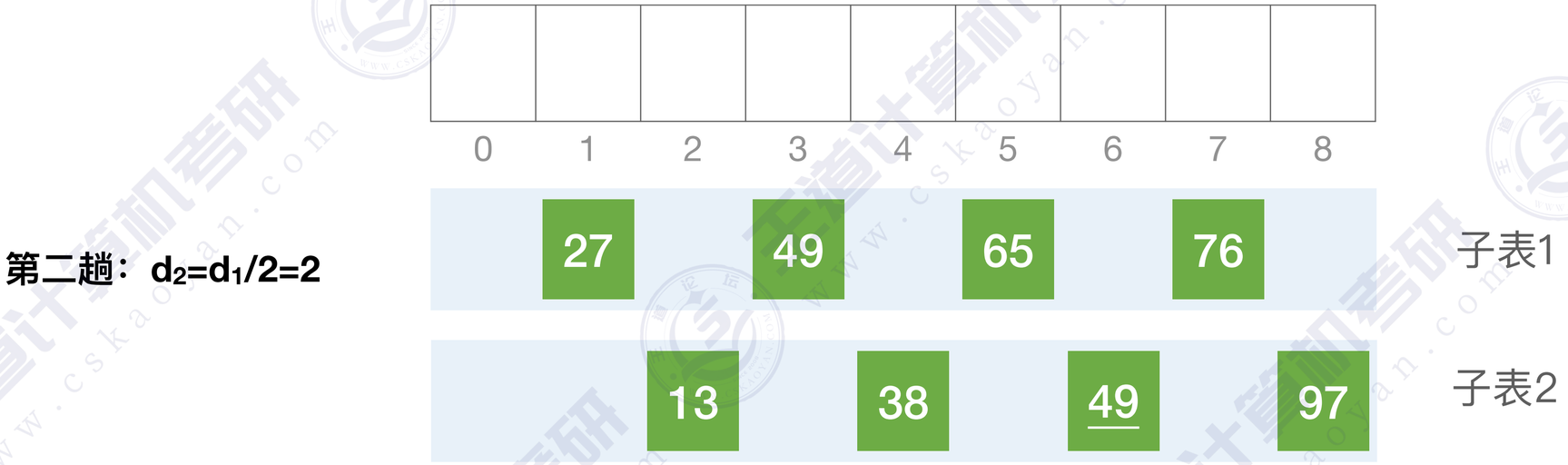
希尔排序：先将待排序表分割成若干形如 $L[i, i+d, i+2d, \ldots, i+kd]$ 的“特殊”子表，对各个子表分别进行直接插入排序。缩小增量d，重复上述过程，直到d=1为止。



第二趟：$d_2=d_1/2=2$

希尔排序（Shell Sort）

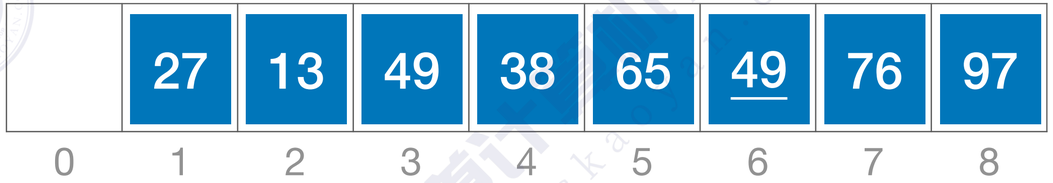
希尔排序：先将待排序表分割成若干形如 $L[i, i+d, i+2d, \ldots, i+kd]$ 的“特殊”子表，对各个子表分别进行直接插入排序。缩小增量d，重复上述过程，直到d=1为止。



第二趟：$d_2=d_1/2=2$

希尔排序（Shell Sort）

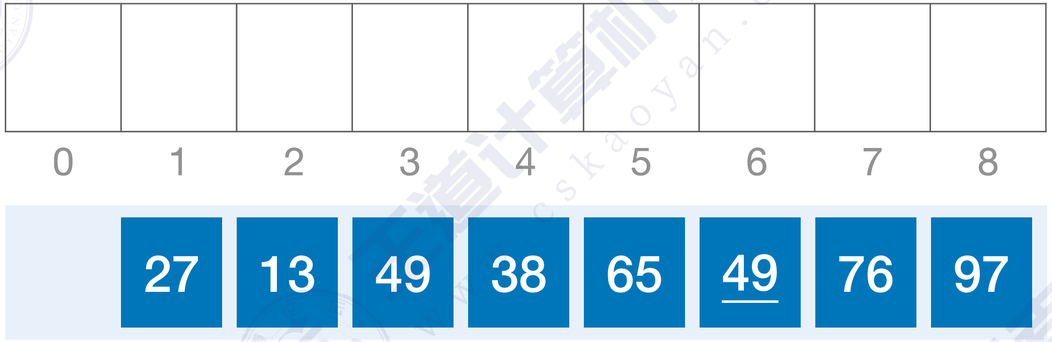
希尔排序：先将待排序表分割成若干形如 $L[i, i+d, i+2d, \ldots, i+kd]$ 的“特殊”子表，对各个子表分别进行直接插入排序。缩小增量d，重复上述过程，直到d=1为止。



第三趟：$d_3=d_2/2=1$

希尔排序（Shell Sort）

希尔排序：先将待排序表分割成若干形如 $L[i, i+d, i+2d, \ldots, i+kd]$ 的“特殊”子表，对各个子表分别进行直接插入排序。缩小增量d，重复上述过程，直到d=1为止。



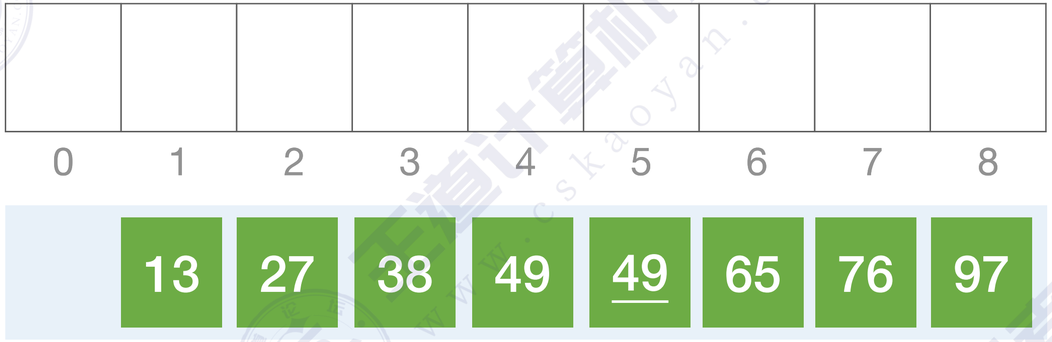
第三趟：$d_3=d_2/2=1$

子表1

整个表已呈现出“基本有序”，对整体再进行一次“直接插入排序”

希尔排序（Shell Sort）

希尔排序：先将待排序表分割成若干形如 $L[i, i+d, i+2d, \ldots, i+kd]$ 的“特殊”子表，对各个子表分别进行直接插入排序。缩小增量d，重复上述过程，直到d=1为止。

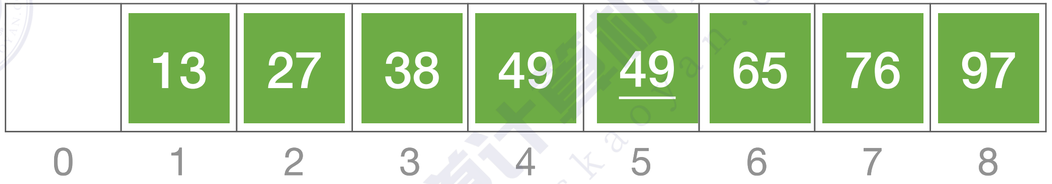


第三趟：$d_3=d_2/2=1$

整个表已呈现出“基本有序”，对整体再进行一次“直接插入排序”

希尔排序（Shell Sort）

希尔排序：先将待排序表分割成若干形如 $L[i, i+d, i+2d, \ldots, i+kd]$ 的“特殊”子表，对各个子表分别进行直接插入排序。缩小增量d，重复上述过程，直到d=1为止。

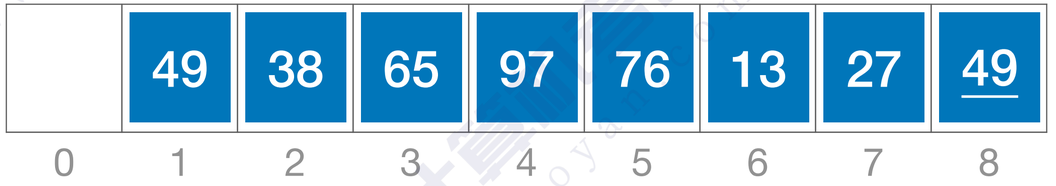


希尔排序（Shell Sort）

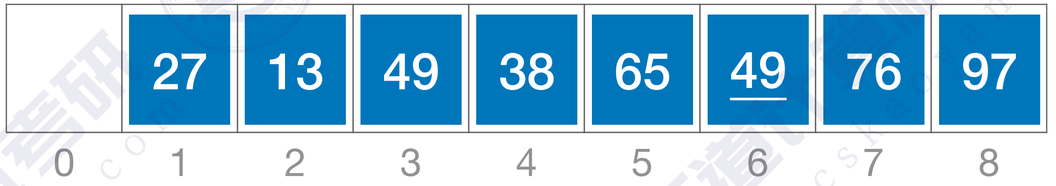
希尔排序：先将待排序表分割成若干形如 $L[i, i+d, i+2d, \ldots, i+kd]$ 的“特殊”子表，对各个子表分别进行直接插入排序。缩小增量d，重复上述过程，直到d=1为止。

希尔本人建议：每次将增量缩小一半

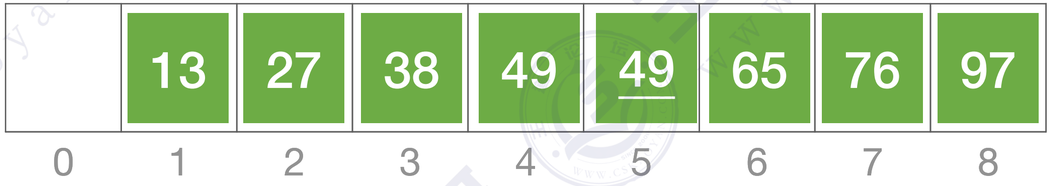
第一趟：$d_1=n/2=4$



第二趟：$d_2=d_1/2=2$

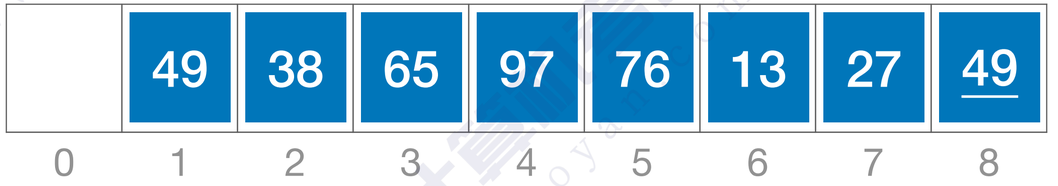


第三趟：$d_3=d_2/2=1$

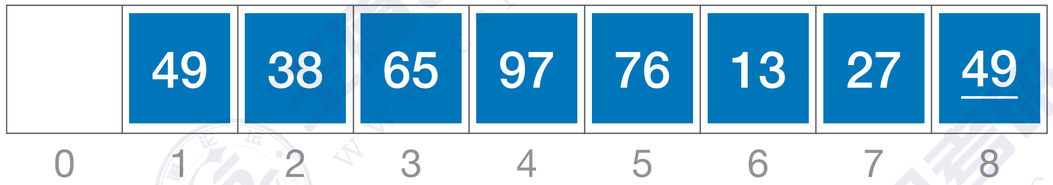


希尔排序（Shell Sort）

希尔排序：先将待排序表分割成若干形如 $L[i, i+d, i+2d, \ldots, i+kd]$ 的“特殊”子表，对各个子表分别进行直接插入排序。缩小增量d，重复上述过程，直到d=1为止。



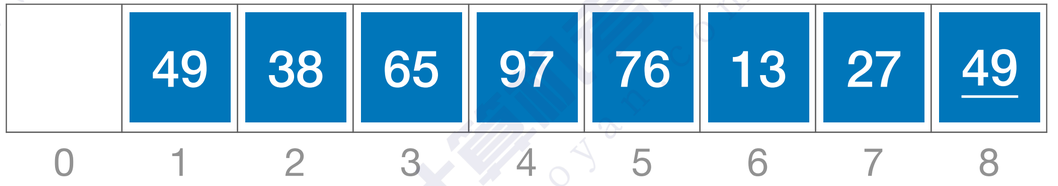
第一趟：$d_1=3$



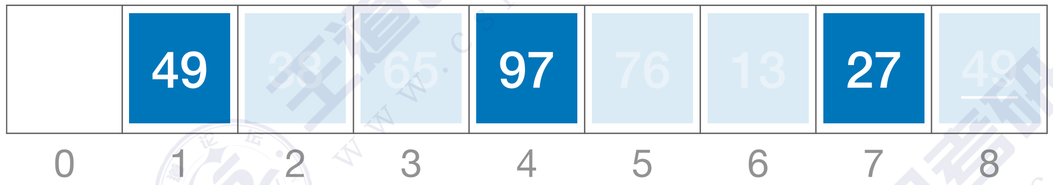
第二趟：$d_2=1$

希尔排序（Shell Sort）

希尔排序：先将待排序表分割成若干形如 $L[i, i+d, i+2d, \ldots, i+kd]$ 的“特殊”子表，对各个子表分别进行直接插入排序。缩小增量d，重复上述过程，直到d=1为止。



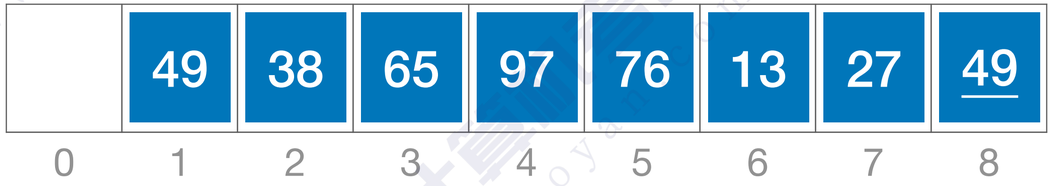
第一趟：$d_1=3$



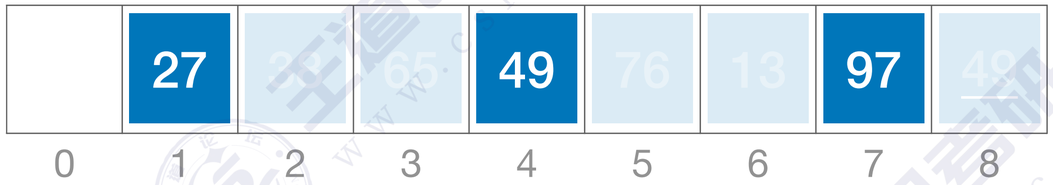
第二趟：$d_2=1$

希尔排序（Shell Sort）

希尔排序：先将待排序表分割成若干形如 $L[i, i+d, i+2d, \ldots, i+kd]$ 的“特殊”子表，对各个子表分别进行直接插入排序。缩小增量d，重复上述过程，直到d=1为止。



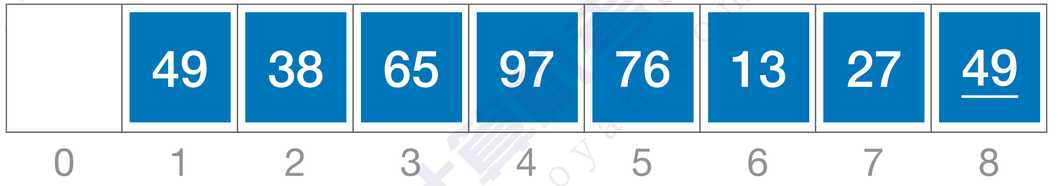
第一趟：$d_1=3$



第二趟：$d_2=1$

希尔排序（Shell Sort）

希尔排序：先将待排序表分割成若干形如 $L[i, i+d, i+2d, \ldots, i+kd]$ 的“特殊”子表，对各个子表分别进行直接插入排序。缩小增量d，重复上述过程，直到d=1为止。

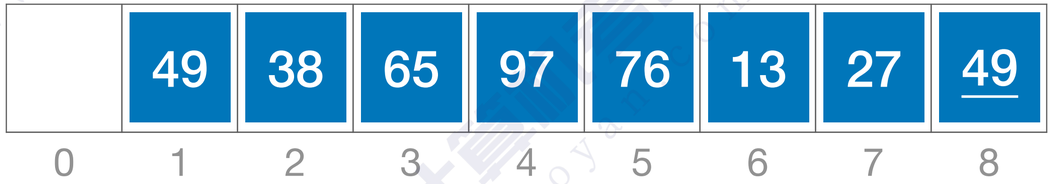


第一趟：$d_1=3$

第二趟：$d_2=1$

希尔排序（Shell Sort）

希尔排序：先将待排序表分割成若干形如 $L[i, i+d, i+2d, \ldots, i+kd]$ 的“特殊”子表，对各个子表分别进行直接插入排序。缩小增量d，重复上述过程，直到d=1为止。

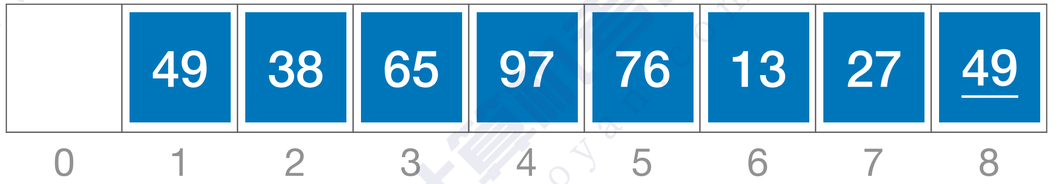


第一趟：$d_1=3$

第二趟：$d_2=1$

希尔排序（Shell Sort）

希尔排序：先将待排序表分割成若干形如 $L[i, i+d, i+2d, \ldots, i+kd]$ 的“特殊”子表，对各个子表分别进行直接插入排序。缩小增量d，重复上述过程，直到d=1为止。

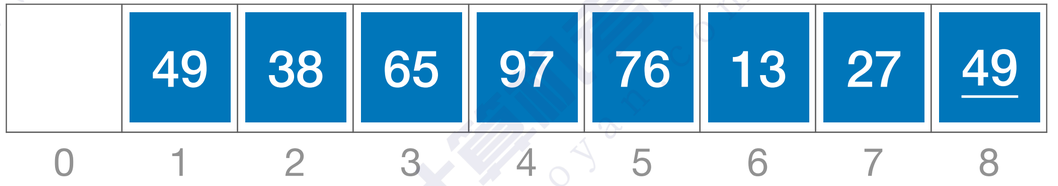


第一趟：$d_1=3$

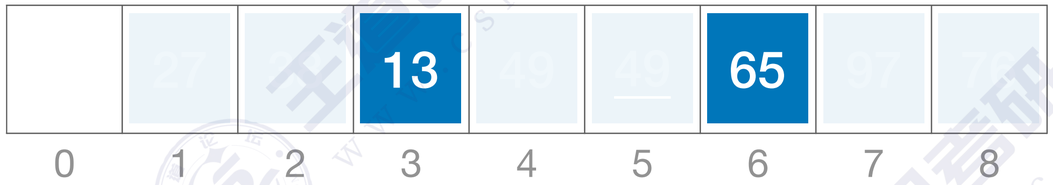
第二趟：$d_2=1$

希尔排序（Shell Sort）

希尔排序：先将待排序表分割成若干形如 $L[i, i+d, i+2d, \ldots, i+kd]$ 的“特殊”子表，对各个子表分别进行直接插入排序。缩小增量d，重复上述过程，直到d=1为止。



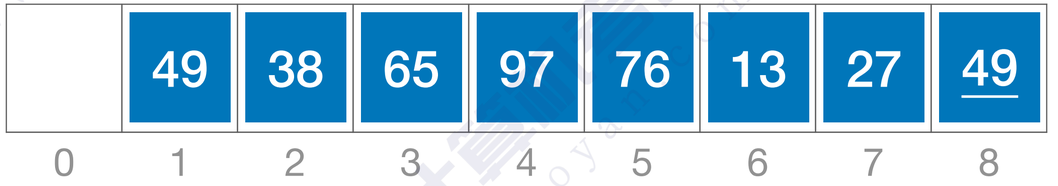
第一趟：$d_1=3$



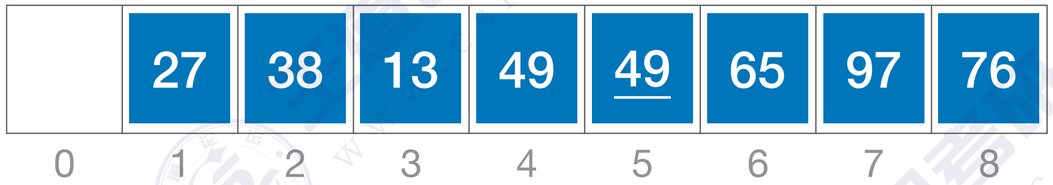
第二趟：$d_2=1$

希尔排序（Shell Sort）

希尔排序：先将待排序表分割成若干形如 $L[i, i+d, i+2d, \ldots, i+kd]$ 的“特殊”子表，对各个子表分别进行直接插入排序。缩小增量d，重复上述过程，直到d=1为止。



第一趟：$d_1=3$



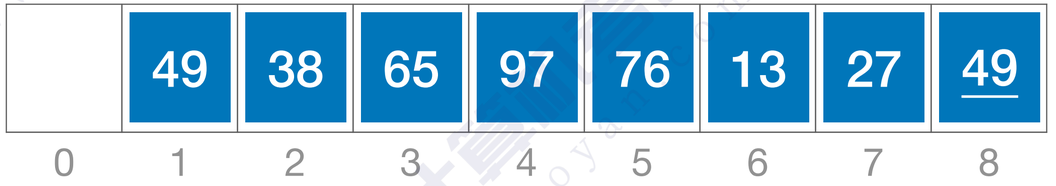
第二趟：$d_2=1$

希尔排序（Shell Sort）

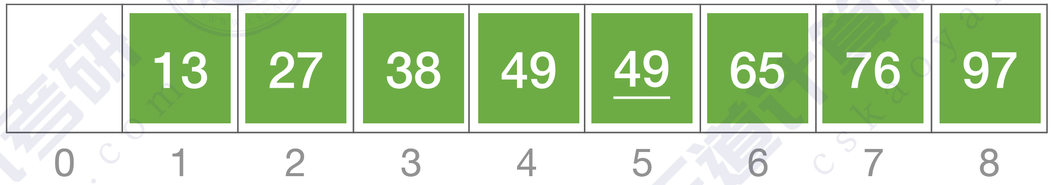
希尔排序：先将待排序表分割成若干形如 $L[i, i+d, i+2d, \ldots, i+kd]$ 的“特殊”子表，对各个子表分别进行直接插入排序。缩小增量d，重复上述过程，直到d=1为止。

考试中可能遇到各种增量

第一趟：$d_1=3$



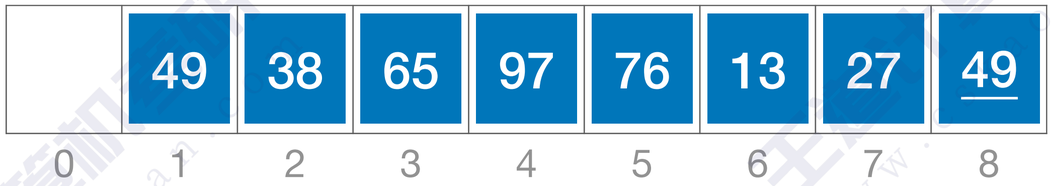
第二趟：$d_2=1$



算法实现

```
//希尔排序
void ShellSort(int A[],int n){
    int d, i, j;
    //A[0]只是暂存单元，不是哨兵，当j<=0时，插入位置已到
    for(d= n/2; d>=1; d=d/2)     //步长变化
        for(i=d+1; i<=n; ++i)
            if(A[i]<A[i-d]){     //需将A[i]插入有序增量子表
                A[0]=A[i];                  //暂存在A[0]
                for(j= i-d; j>0 && A[0]<A[j]; j-=d)
                    A[j+d]=A[j];            //记录后移，查找插入的位置
                A[j+d]=A[0];                //插入
            }//if
}
```

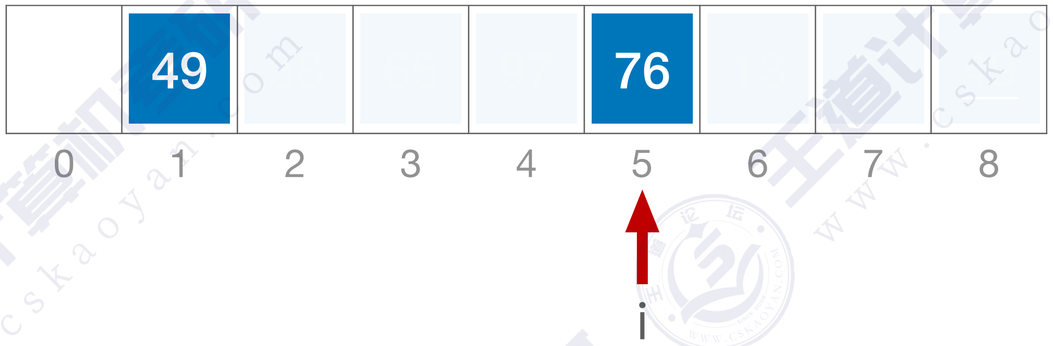
第一趟：d=n/2=4



算法实现

```
//希尔排序
void ShellSort(int A[],int n){
    int d, i, j;
    //A[0]只是暂存单元，不是哨兵，当j<=0时，插入位置已到
    for(d= n/2; d>=1; d=d/2)     //步长变化
        for(i=d+1; i<=n; ++i)
            if(A[i]<A[i-d]){     //需将A[i]插入有序增量子表
                A[0]=A[i];                  //暂存在A[0]
                for(j= i-d; j>0 && A[0]<A[j]; j-=d)
                    A[j+d]=A[j];            //记录后移，查找插入的位置
                A[j+d]=A[0];                //插入
            }//if
}
```

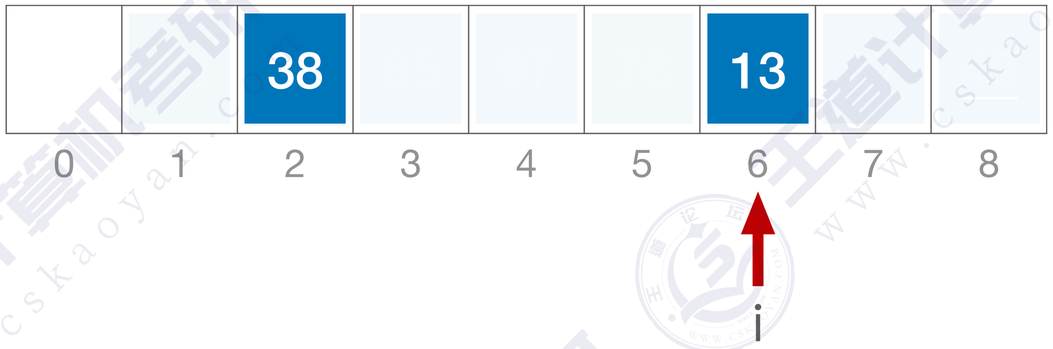
第一趟：d=n/2=4



算法实现

```
//希尔排序
void ShellSort(int A[],int n){
    int d, i, j;
    //A[0]只是暂存单元，不是哨兵，当j<=0时，插入位置已到
    for(d= n/2; d>=1; d=d/2)     //步长变化
        for(i=d+1; i<=n; ++i)
            if(A[i]<A[i-d]){     //需将A[i]插入有序增量子表
                A[0]=A[i];                  //暂存在A[0]
                for(j= i-d; j>0 && A[0]<A[j]; j-=d)
                    A[j+d]=A[j];            //记录后移，查找插入的位置
                A[j+d]=A[0];                //插入
            }//if
}
```

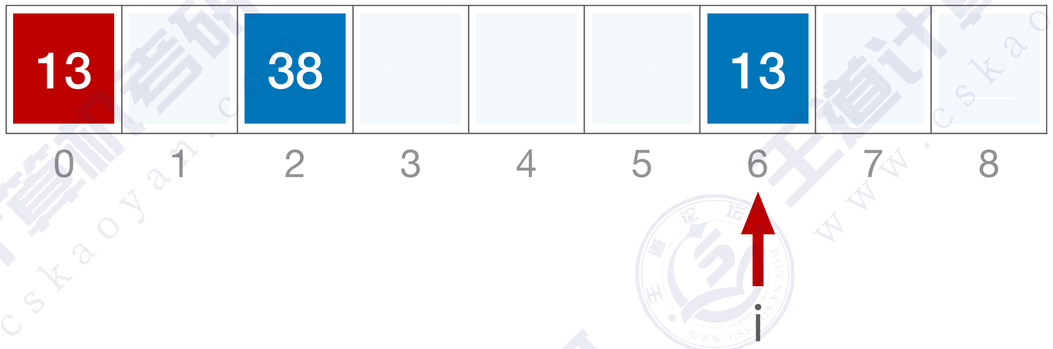
第一趟：d=n/2=4



算法实现

```
//希尔排序
void ShellSort(int A[],int n){
    int d, i, j;
    //A[0]只是暂存单元，不是哨兵，当j<=0时，插入位置已到
    for(d= n/2; d>=1; d=d/2)     //步长变化
        for(i=d+1; i<=n; ++i)
            if(A[i]<A[i-d]){     //需将A[i]插入有序增量子表
                A[0]=A[i];                  //暂存在A[0]
                for(j= i-d; j>0 && A[0]<A[j]; j-=d)
                    A[j+d]=A[j];            //记录后移，查找插入的位置
                A[j+d]=A[0];                //插入
            }//if
}
```

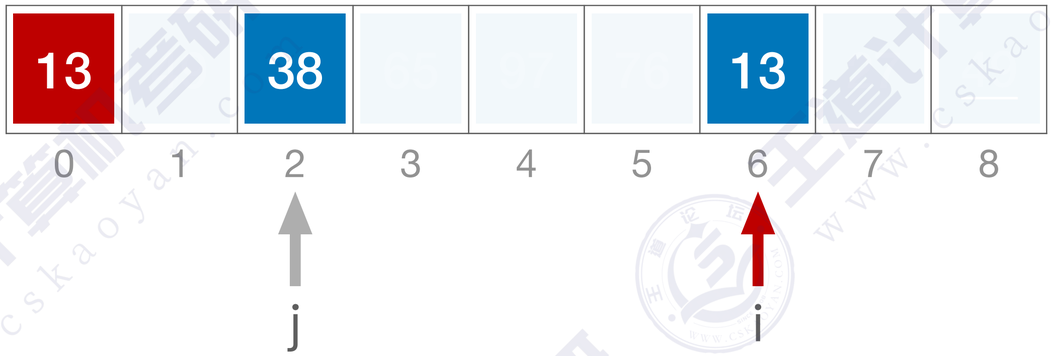
第一趟：d=n/2=4



算法实现

```
//希尔排序
void ShellSort(int A[],int n){
    int d, i, j;
    //A[0]只是暂存单元，不是哨兵，当j<=0时，插入位置已到
    for(d= n/2; d>=1; d=d/2)     //步长变化
        for(i=d+1; i<=n; ++i)
            if(A[i]<A[i-d]){     //需将A[i]插入有序增量子表
                A[0]=A[i];                  //暂存在A[0]
                for(j= i-d; j>0 && A[0]<A[j]; j-=d)
                    A[j+d]=A[j];            //记录后移，查找插入的位置
                A[j+d]=A[0];                //插入
            }//if
}
```

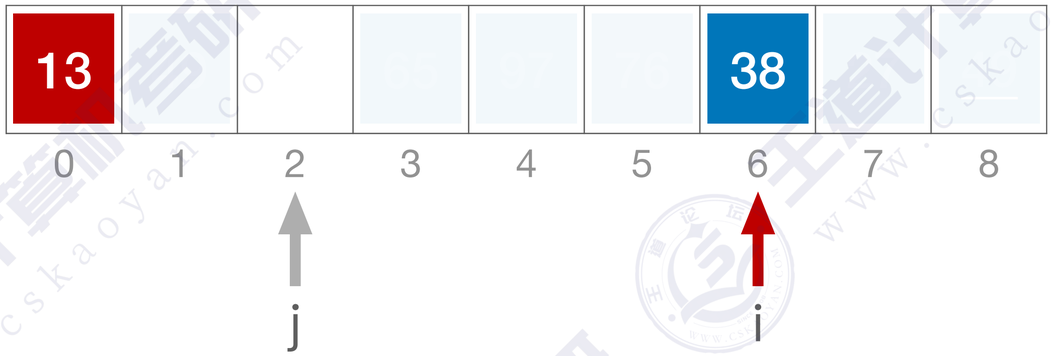
第一趟：d=n/2=4



算法实现

```
//希尔排序
void ShellSort(int A[],int n){
    int d, i, j;
    //A[0]只是暂存单元，不是哨兵，当j<=0时，插入位置已到
    for(d= n/2; d>=1; d=d/2)     //步长变化
        for(i=d+1; i<=n; ++i)
            if(A[i]<A[i-d]){     //需将A[i]插入有序增量子表
                A[0]=A[i];                  //暂存在A[0]
                for(j= i-d; j>0 && A[0]<A[j]; j-=d)
                    A[j+d]=A[j];            //记录后移，查找插入的位置
                A[j+d]=A[0];                //插入
            }//if
}
```

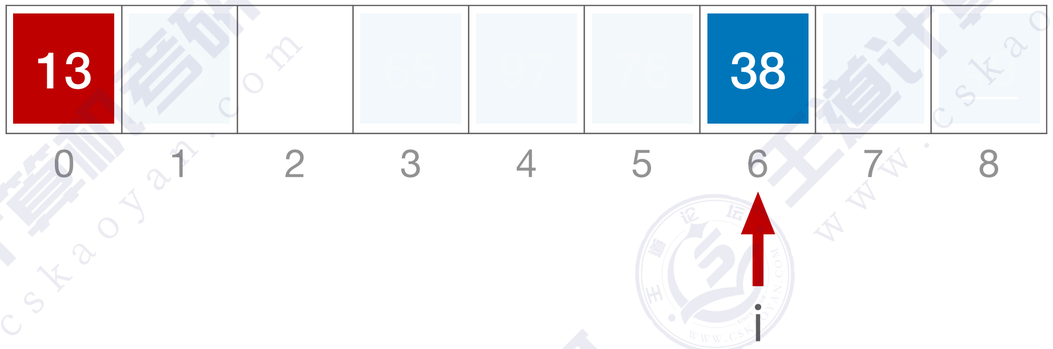
第一趟：d=n/2=4



算法实现

```
//希尔排序
void ShellSort(int A[],int n){
    int d, i, j;
    //A[0]只是暂存单元，不是哨兵，当j<=0时，插入位置已到
    for(d= n/2; d>=1; d=d/2)     //步长变化
        for(i=d+1; i<=n; ++i)
            if(A[i]<A[i-d]){     //需将A[i]插入有序增量子表
                A[0]=A[i];                  //暂存在A[0]
                for(j= i-d; j>0 && A[0]<A[j]; j-=d)
                    A[j+d]=A[j];            //记录后移，查找插入的位置
                A[j+d]=A[0];                //插入
            }//if
}
```

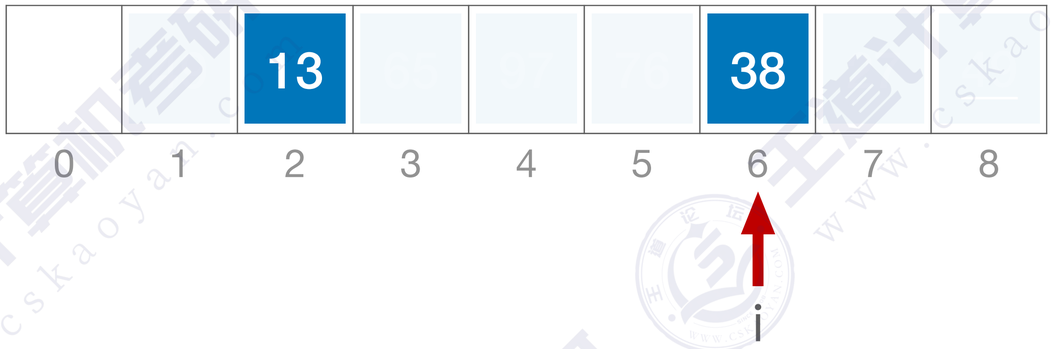
第一趟：d=n/2=4



算法实现

```
//希尔排序
void ShellSort(int A[],int n){
    int d, i, j;
    //A[0]只是暂存单元，不是哨兵，当j<=0时，插入位置已到
    for(d= n/2; d>=1; d=d/2)     //步长变化
        for(i=d+1; i<=n; ++i)
            if(A[i]<A[i-d]){     //需将A[i]插入有序增量子表
                A[0]=A[i];                  //暂存在A[0]
                for(j= i-d; j>0 && A[0]<A[j]; j-=d)
                    A[j+d]=A[j];            //记录后移，查找插入的位置
                A[j+d]=A[0];                //插入
            }//if
}
```

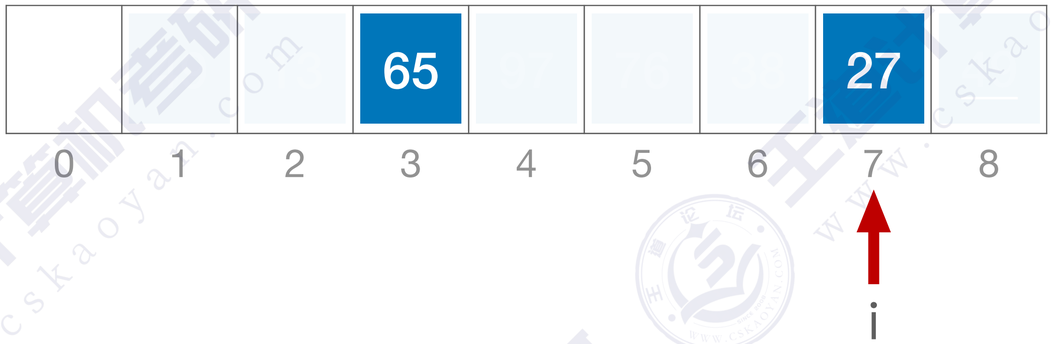
第一趟：d=n/2=4



算法实现

```
//希尔排序
void ShellSort(int A[],int n){
    int d, i, j;
    //A[0]只是暂存单元，不是哨兵，当j<=0时，插入位置已到
    for(d= n/2; d>=1; d=d/2)     //步长变化
        for(i=d+1; i<=n; ++i)
            if(A[i]<A[i-d]){     //需将A[i]插入有序增量子表
                A[0]=A[i];                  //暂存在A[0]
                for(j= i-d; j>0 && A[0]<A[j]; j-=d)
                    A[j+d]=A[j];            //记录后移，查找插入的位置
                A[j+d]=A[0];                //插入
            }//if
}
```

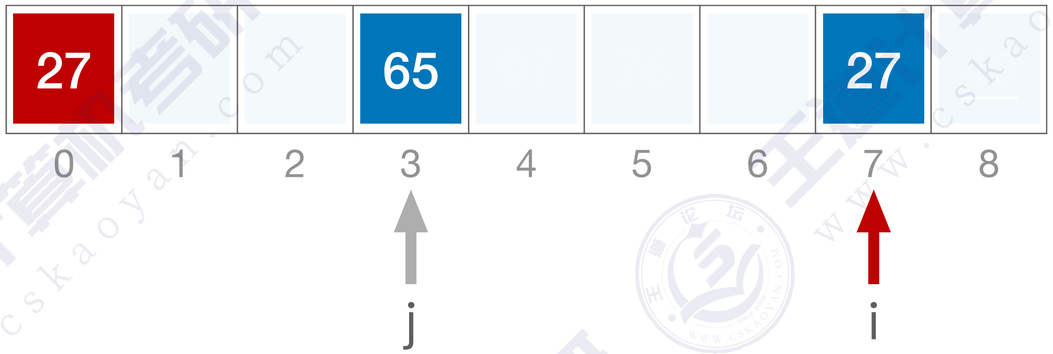
第一趟：d=n/2=4



算法实现

```
//希尔排序
void ShellSort(int A[],int n){
    int d, i, j;
    //A[0]只是暂存单元，不是哨兵，当j<=0时，插入位置已到
    for(d= n/2; d>=1; d=d/2)     //步长变化
        for(i=d+1; i<=n; ++i)
            if(A[i]<A[i-d]){     //需将A[i]插入有序增量子表
                A[0]=A[i];                  //暂存在A[0]
                for(j= i-d; j>0 && A[0]<A[j]; j-=d)
                    A[j+d]=A[j];            //记录后移，查找插入的位置
                A[j+d]=A[0];                //插入
            }//if
}
```

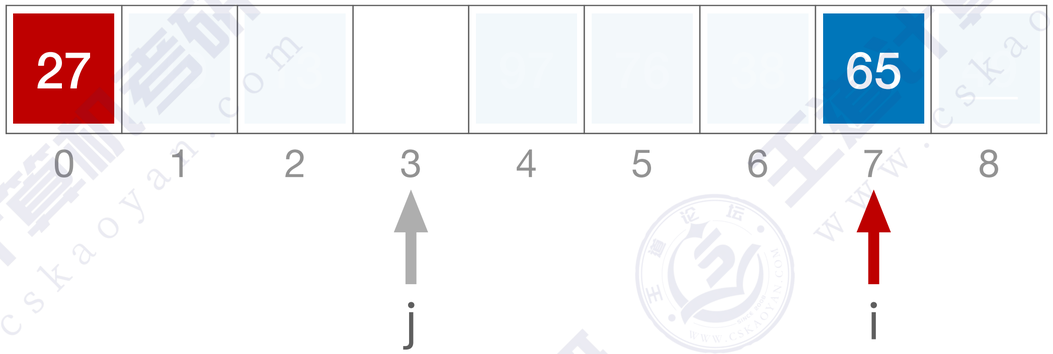
第一趟：d=n/2=4



算法实现

```
//希尔排序
void ShellSort(int A[],int n){
    int d, i, j;
    //A[0]只是暂存单元，不是哨兵，当j<=0时，插入位置已到
    for(d= n/2; d>=1; d=d/2)     //步长变化
        for(i=d+1; i<=n; ++i)
            if(A[i]<A[i-d]){     //需将A[i]插入有序增量子表
                A[0]=A[i];                  //暂存在A[0]
                for(j= i-d; j>0 && A[0]<A[j]; j-=d)
                    A[j+d]=A[j];            //记录后移，查找插入的位置
                A[j+d]=A[0];                //插入
            }//if
}
```

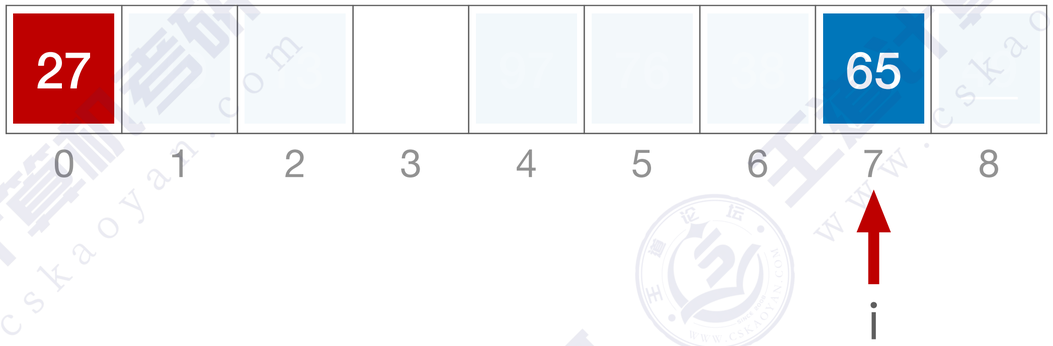
第一趟：d=n/2=4



算法实现

```
//希尔排序
void ShellSort(int A[],int n){
    int d, i, j;
    //A[0]只是暂存单元，不是哨兵，当j<=0时，插入位置已到
    for(d= n/2; d>=1; d=d/2)     //步长变化
        for(i=d+1; i<=n; ++i)
            if(A[i]<A[i-d]){     //需将A[i]插入有序增量子表
                A[0]=A[i];                  //暂存在A[0]
                for(j= i-d; j>0 && A[0]<A[j]; j-=d)
                    A[j+d]=A[j];            //记录后移，查找插入的位置
                A[j+d]=A[0];                //插入
            }//if
}
```

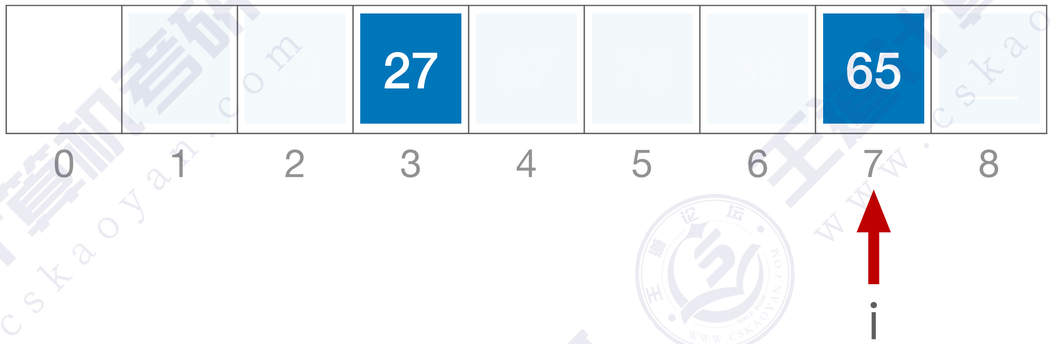
第一趟：d=n/2=4



算法实现

```
//希尔排序
void ShellSort(int A[],int n){
    int d, i, j;
    //A[0]只是暂存单元，不是哨兵，当j<=0时，插入位置已到
    for(d= n/2; d>=1; d=d/2)     //步长变化
        for(i=d+1; i<=n; ++i)
            if(A[i]<A[i-d]){     //需将A[i]插入有序增量子表
                A[0]=A[i];                  //暂存在A[0]
                for(j= i-d; j>0 && A[0]<A[j]; j-=d)
                    A[j+d]=A[j];            //记录后移，查找插入的位置
                A[j+d]=A[0];                //插入
            }//if
}
```

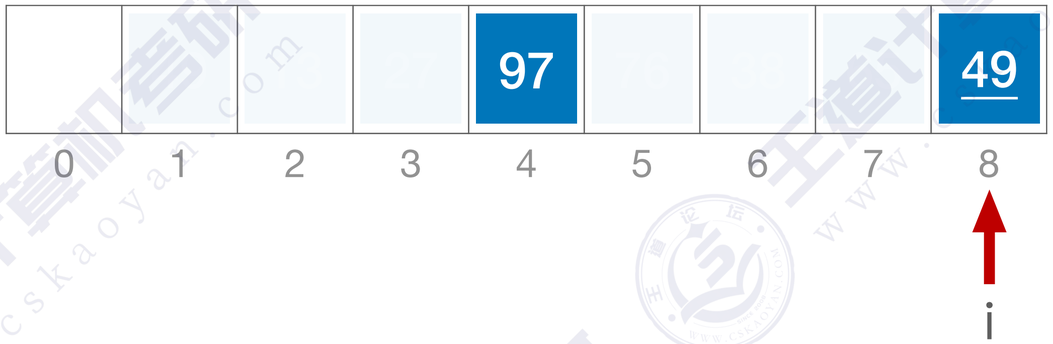
第一趟：d=n/2=4



算法实现

```
//希尔排序
void ShellSort(int A[],int n){
    int d, i, j;
    //A[0]只是暂存单元，不是哨兵，当j<=0时，插入位置已到
    for(d= n/2; d>=1; d=d/2)     //步长变化
        for(i=d+1; i<=n; ++i)
            if(A[i]<A[i-d]){     //需将A[i]插入有序增量子表
                A[0]=A[i];                  //暂存在A[0]
                for(j= i-d; j>0 && A[0]<A[j]; j-=d)
                    A[j+d]=A[j];            //记录后移，查找插入的位置
                A[j+d]=A[0];                //插入
            }//if
}
```

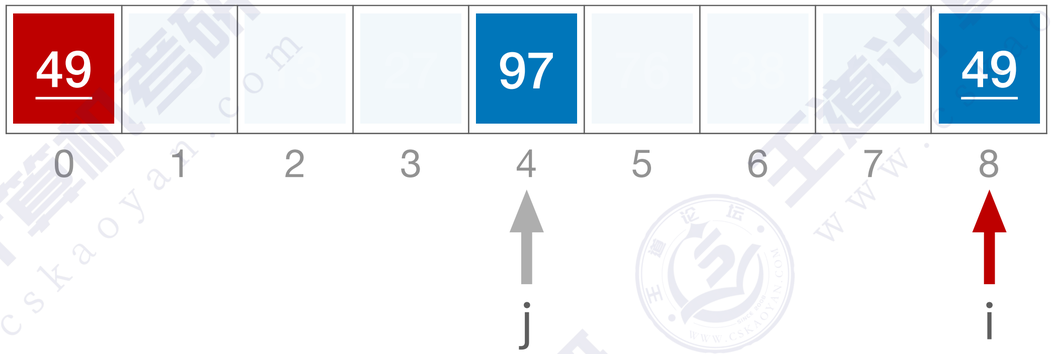
第一趟：d=n/2=4



算法实现

```
//希尔排序
void ShellSort(int A[],int n){
    int d, i, j;
    //A[0]只是暂存单元，不是哨兵，当j<=0时，插入位置已到
    for(d= n/2; d>=1; d=d/2)     //步长变化
        for(i=d+1; i<=n; ++i)
            if(A[i]<A[i-d]){     //需将A[i]插入有序增量子表
                A[0]=A[i];                  //暂存在A[0]
                for(j= i-d; j>0 && A[0]<A[j]; j-=d)
                    A[j+d]=A[j];            //记录后移，查找插入的位置
                A[j+d]=A[0];                //插入
            }//if
}
```

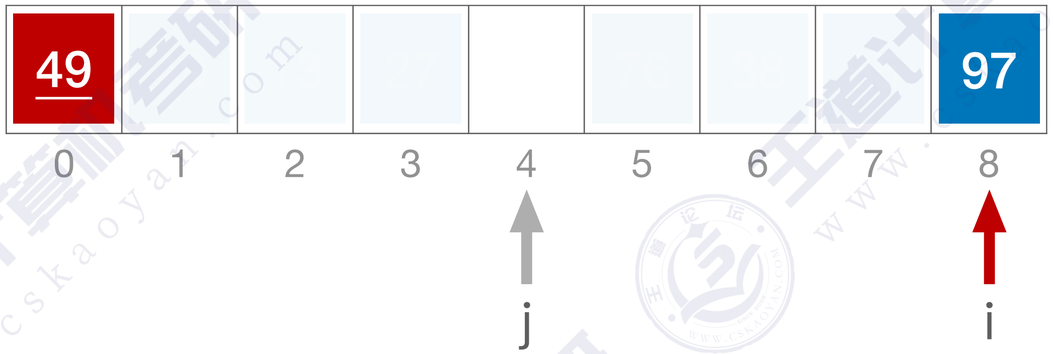
第一趟：d=n/2=4



算法实现

```
//希尔排序
void ShellSort(int A[],int n){
    int d, i, j;
    //A[0]只是暂存单元，不是哨兵，当j<=0时，插入位置已到
    for(d= n/2; d>=1; d=d/2)     //步长变化
        for(i=d+1; i<=n; ++i)
            if(A[i]<A[i-d]){     //需将A[i]插入有序增量子表
                A[0]=A[i];                  //暂存在A[0]
                for(j= i-d; j>0 && A[0]<A[j]; j-=d)
                    A[j+d]=A[j];            //记录后移，查找插入的位置
                A[j+d]=A[0];                //插入
            }//if
}
```

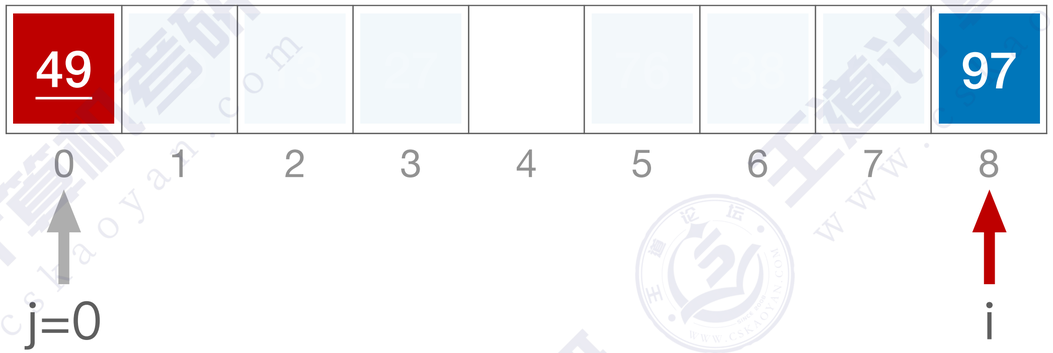
第一趟：d=n/2=4



算法实现

```
//希尔排序
void ShellSort(int A[],int n){
    int d, i, j;
    //A[0]只是暂存单元，不是哨兵，当j<=0时，插入位置已到
    for(d= n/2; d>=1; d=d/2)     //步长变化
        for(i=d+1; i<=n; ++i)
            if(A[i]<A[i-d]){     //需将A[i]插入有序增量子表
                A[0]=A[i];                  //暂存在A[0]
                for(j= i-d; j>0 && A[0]<A[j]; j-=d)
                    A[j+d]=A[j];            //记录后移，查找插入的位置
                A[j+d]=A[0];                //插入
            }//if
}
```

第一趟：d=n/2=4



算法实现

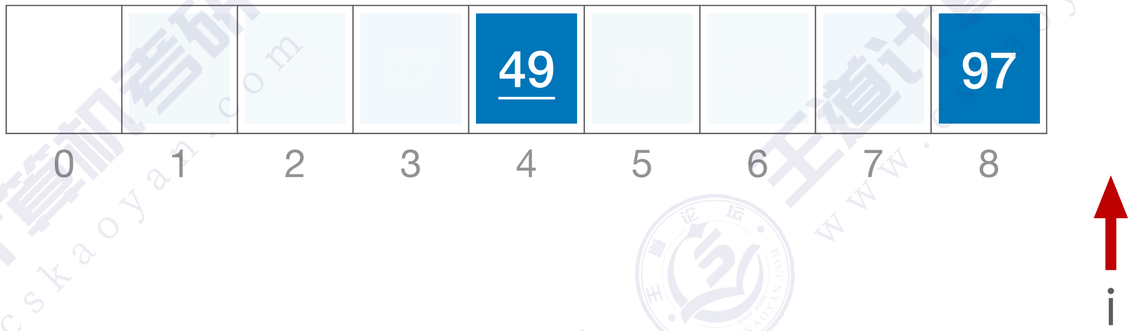
```
//希尔排序
void ShellSort(int A[],int n){
    int d, i, j;
    //A[0]只是暂存单元，不是哨兵，当j<=0时，插入位置已到
    for(d= n/2; d>=1; d=d/2)     //步长变化
        for(i=d+1; i<=n; ++i)
            if(A[i]<A[i-d]){     //需将A[i]插入有序增量子表
                A[0]=A[i];                  //暂存在A[0]
                for(j= i-d; j>0 && A[0]<A[j]; j-=d)
                    A[j+d]=A[j];            //记录后移，查找插入的位置
                A[j+d]=A[0];                //插入
            }//if
}
```

第一趟：d=n/2=4

算法实现

```
//希尔排序
void ShellSort(int A[],int n){
    int d, i, j;
    //A[0]只是暂存单元，不是哨兵，当j<=0时，插入位置已到
    for(d= n/2; d>=1; d=d/2)     //步长变化
        for(i=d+1; i<=n; ++i)
            if(A[i]<A[i-d]){     //需将A[i]插入有序增量子表
                A[0]=A[i];                  //暂存在A[0]
                for(j= i-d; j>0 && A[0]<A[j]; j-=d)
                    A[j+d]=A[j];            //记录后移，查找插入的位置
                A[j+d]=A[0];                //插入
            }//if
}
```

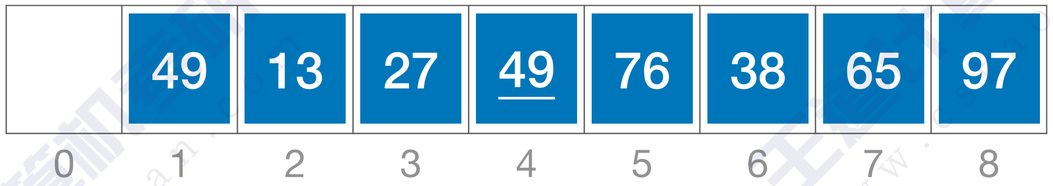
第一趟：d=n/2=4



算法实现

```
//希尔排序
void ShellSort(int A[],int n){
    int d, i, j;
    //A[0]只是暂存单元，不是哨兵，当j<=0时，插入位置已到
    for(d= n/2; d>=1; d=d/2)     //步长变化
        for(i=d+1; i<=n; ++i)
            if(A[i]<A[i-d]){     //需将A[i]插入有序增量子表
                A[0]=A[i];                  //暂存在A[0]
                for(j= i-d; j>0 && A[0]<A[j]; j-=d)
                    A[j+d]=A[j];            //记录后移，查找插入的位置
                A[j+d]=A[0];                //插入
            }//if
}
```

第二趟：d=2



算法实现

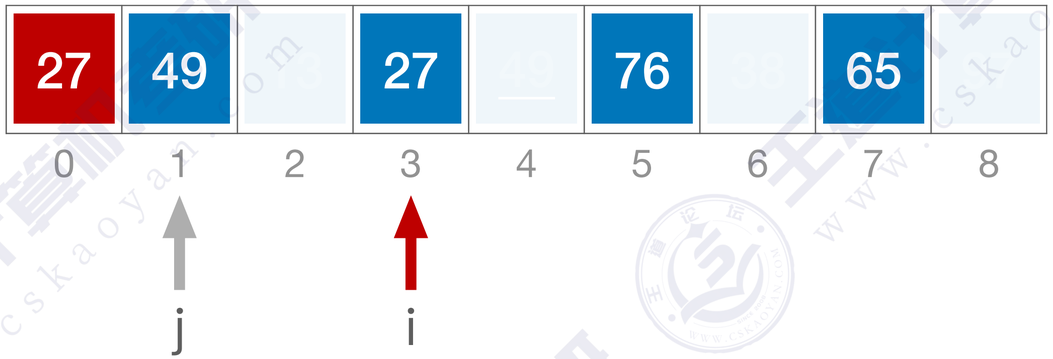
```
//希尔排序
void ShellSort(int A[],int n){
    int d, i, j;
    //A[0]只是暂存单元，不是哨兵，当j<=0时，插入位置已到
    for(d= n/2; d>=1; d=d/2)     //步长变化
        for(i=d+1; i<=n; ++i)
            if(A[i]<A[i-d]){     //需将A[i]插入有序增量子表
                A[0]=A[i];                  //暂存在A[0]
                for(j= i-d; j>0 && A[0]<A[j]; j-=d)
                    A[j+d]=A[j];            //记录后移，查找插入的位置
                A[j+d]=A[0];                //插入
            }//if
}
```

第二趟：d=2

算法实现

```
//希尔排序
void ShellSort(int A[],int n){
    int d, i, j;
    //A[0]只是暂存单元，不是哨兵，当j<=0时，插入位置已到
    for(d= n/2; d>=1; d=d/2)     //步长变化
        for(i=d+1; i<=n; ++i)
            if(A[i]<A[i-d]){     //需将A[i]插入有序增量子表
                A[0]=A[i];                  //暂存在A[0]
                for(j= i-d; j>0 && A[0]<A[j]; j-=d)
                    A[j+d]=A[j];            //记录后移，查找插入的位置
                A[j+d]=A[0];                //插入
            }//if
}
```

第二趟：d=2



算法实现

```
//希尔排序
void ShellSort(int A[],int n){
    int d, i, j;
    //A[0]只是暂存单元，不是哨兵，当j<=0时，插入位置已到
    for(d= n/2; d>=1; d=d/2)     //步长变化
        for(i=d+1; i<=n; ++i)
            if(A[i]<A[i-d]){     //需将A[i]插入有序增量子表
                A[0]=A[i];                  //暂存在A[0]
                for(j= i-d; j>0 && A[0]<A[j]; j-=d)
                    A[j+d]=A[j];            //记录后移，查找插入的位置
                A[j+d]=A[0];                //插入
            }//if
}
```

第二趟：d=2

算法实现

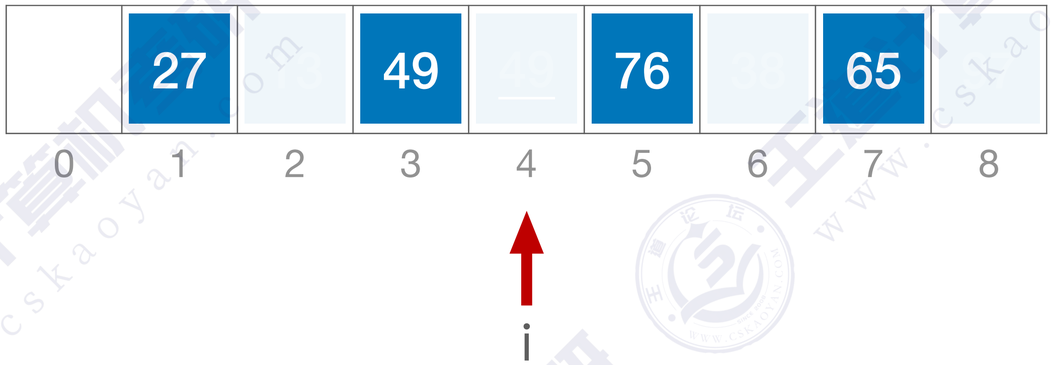
```
//希尔排序
void ShellSort(int A[],int n){
    int d, i, j;
    //A[0]只是暂存单元，不是哨兵，当j<=0时，插入位置已到
    for(d= n/2; d>=1; d=d/2)     //步长变化
        for(i=d+1; i<=n; ++i)
            if(A[i]<A[i-d]){     //需将A[i]插入有序增量子表
                A[0]=A[i];                  //暂存在A[0]
                for(j= i-d; j>0 && A[0]<A[j]; j-=d)
                    A[j+d]=A[j];            //记录后移，查找插入的位置
                A[j+d]=A[0];                //插入
            }//if
}
```

第二趟：d=2

算法实现

```
//希尔排序
void ShellSort(int A[],int n){
    int d, i, j;
    //A[0]只是暂存单元，不是哨兵，当j<=0时，插入位置已到
    for(d= n/2; d>=1; d=d/2)     //步长变化
        for(i=d+1; i<=n; ++i)
            if(A[i]<A[i-d]){     //需将A[i]插入有序增量子表
                A[0]=A[i];                  //暂存在A[0]
                for(j= i-d; j>0 && A[0]<A[j]; j-=d)
                    A[j+d]=A[j];            //记录后移，查找插入的位置
                A[j+d]=A[0];                //插入
            }//if
}
```

第二趟：d=2



算法实现

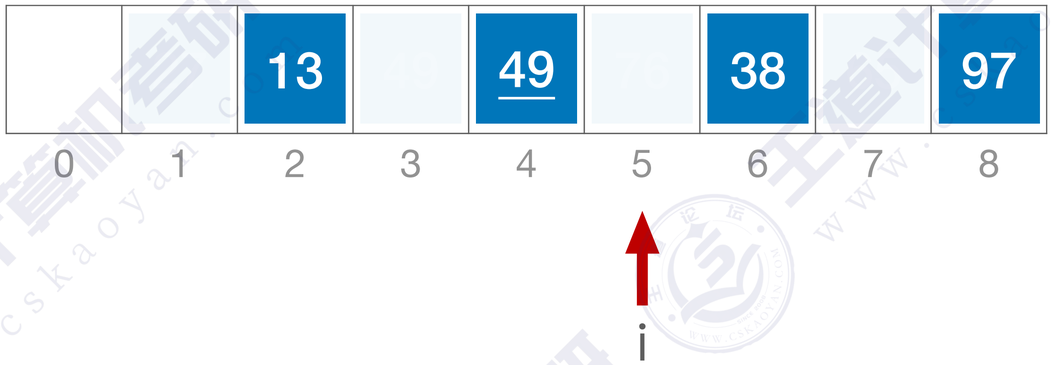
```
//希尔排序
void ShellSort(int A[],int n){
    int d, i, j;
    //A[0]只是暂存单元，不是哨兵，当j<=0时，插入位置已到
    for(d= n/2; d>=1; d=d/2)     //步长变化
        for(i=d+1; i<=n; ++i)
            if(A[i]<A[i-d]){     //需将A[i]插入有序增量子表
                A[0]=A[i];                  //暂存在A[0]
                for(j= i-d; j>0 && A[0]<A[j]; j-=d)
                    A[j+d]=A[j];            //记录后移，查找插入的位置
                A[j+d]=A[0];                //插入
            }//if
}
```

第二趟：d=2

算法实现

```
//希尔排序
void ShellSort(int A[],int n){
    int d, i, j;
    //A[0]只是暂存单元，不是哨兵，当j<=0时，插入位置已到
    for(d= n/2; d>=1; d=d/2)     //步长变化
        for(i=d+1; i<=n; ++i)
            if(A[i]<A[i-d]){     //需将A[i]插入有序增量子表
                A[0]=A[i];                  //暂存在A[0]
                for(j= i-d; j>0 && A[0]<A[j]; j-=d)
                    A[j+d]=A[j];            //记录后移，查找插入的位置
                A[j+d]=A[0];                //插入
            }//if
}
```

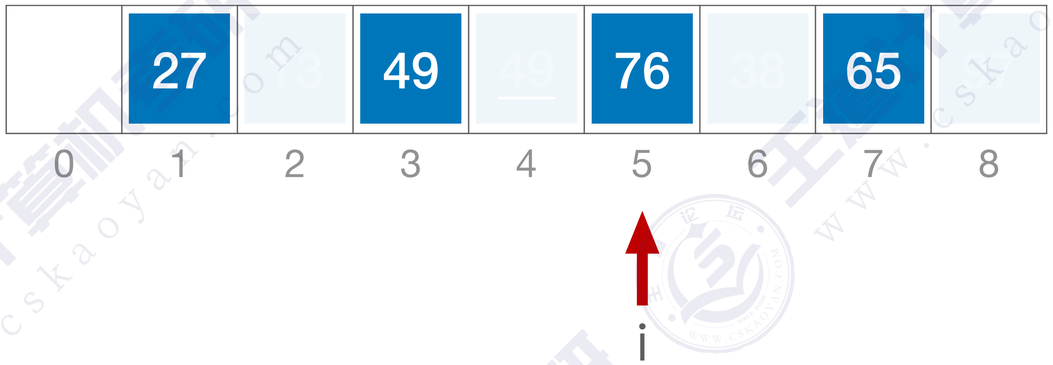
第二趟：d=2



算法实现

```
//希尔排序
void ShellSort(int A[],int n){
    int d, i, j;
    //A[0]只是暂存单元，不是哨兵，当j<=0时，插入位置已到
    for(d= n/2; d>=1; d=d/2)     //步长变化
        for(i=d+1; i<=n; ++i)
            if(A[i]<A[i-d]){     //需将A[i]插入有序增量子表
                A[0]=A[i];                  //暂存在A[0]
                for(j= i-d; j>0 && A[0]<A[j]; j-=d)
                    A[j+d]=A[j];            //记录后移，查找插入的位置
                A[j+d]=A[0];                //插入
            }//if
}
```

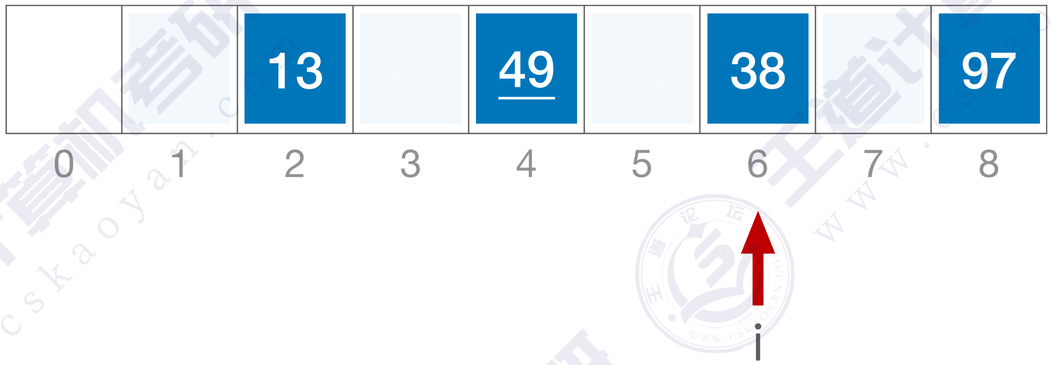
第二趟：d=2



算法实现

```
//希尔排序
void ShellSort(int A[],int n){
    int d, i, j;
    //A[0]只是暂存单元，不是哨兵，当j<=0时，插入位置已到
    for(d= n/2; d>=1; d=d/2)     //步长变化
        for(i=d+1; i<=n; ++i)
            if(A[i]<A[i-d]){     //需将A[i]插入有序增量子表
                A[0]=A[i];                  //暂存在A[0]
                for(j= i-d; j>0 && A[0]<A[j]; j-=d)
                    A[j+d]=A[j];            //记录后移，查找插入的位置
                A[j+d]=A[0];                //插入
            }//if
}
```

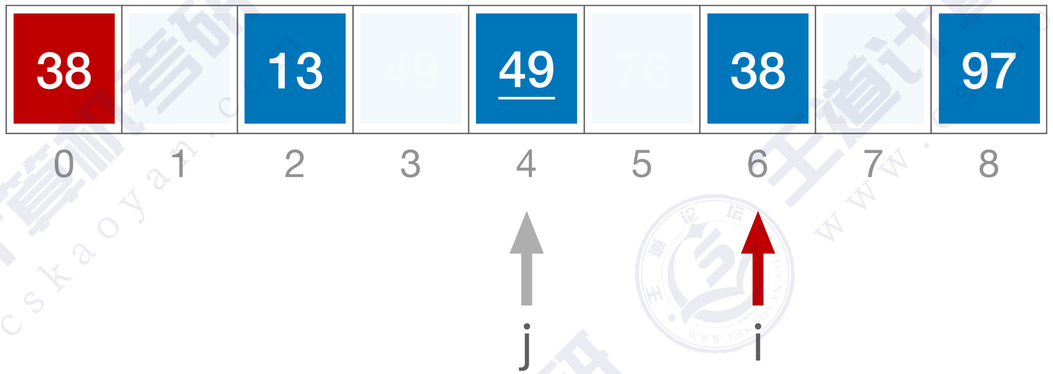
第二趟：d=2



算法实现

```
//希尔排序
void ShellSort(int A[],int n){
    int d, i, j;
    //A[0]只是暂存单元，不是哨兵，当j<=0时，插入位置已到
    for(d= n/2; d>=1; d=d/2)     //步长变化
        for(i=d+1; i<=n; ++i)
            if(A[i]<A[i-d]){     //需将A[i]插入有序增量子表
                A[0]=A[i];                  //暂存在A[0]
                for(j= i-d; j>0 && A[0]<A[j]; j-=d)
                    A[j+d]=A[j];            //记录后移，查找插入的位置
                A[j+d]=A[0];                //插入
            }//if
}
```

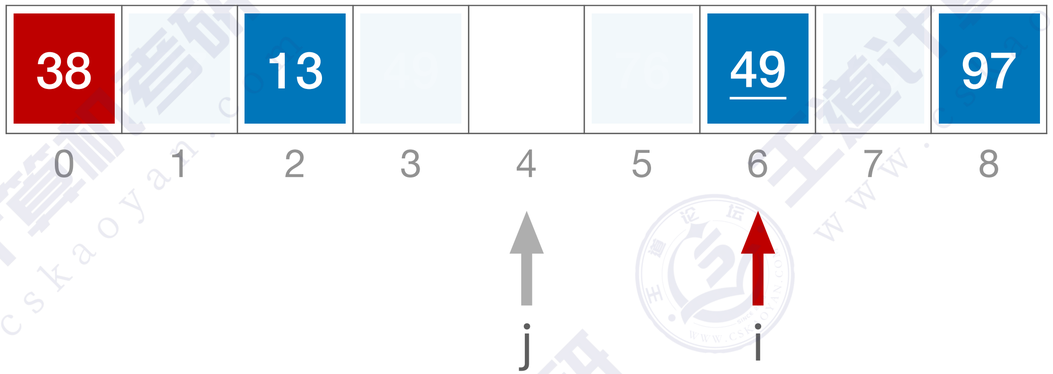
第二趟：d=2



算法实现

```
//希尔排序
void ShellSort(int A[],int n){
    int d, i, j;
    //A[0]只是暂存单元，不是哨兵，当j<=0时，插入位置已到
    for(d= n/2; d>=1; d=d/2)     //步长变化
        for(i=d+1; i<=n; ++i)
            if(A[i]<A[i-d]){     //需将A[i]插入有序增量子表
                A[0]=A[i];                  //暂存在A[0]
                for(j= i-d; j>0 && A[0]<A[j]; j-=d)
                    A[j+d]=A[j];            //记录后移，查找插入的位置
                A[j+d]=A[0];                //插入
            }//if
}
```

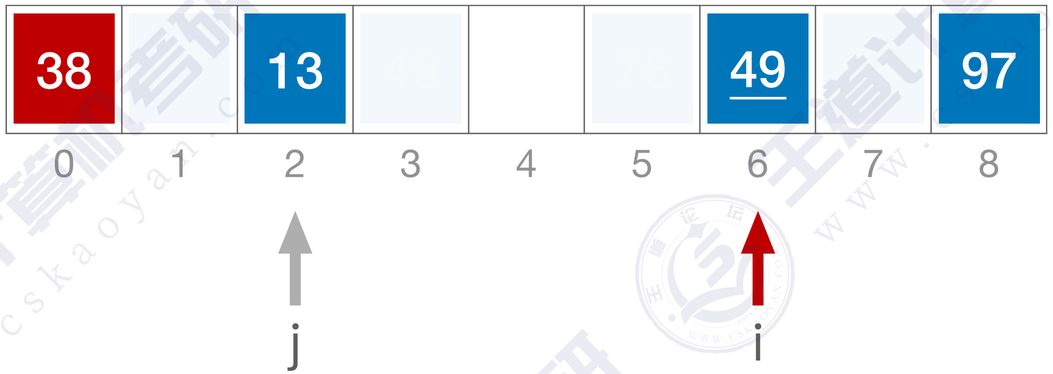
第二趟：d=2



算法实现

```
//希尔排序
void ShellSort(int A[],int n){
    int d, i, j;
    //A[0]只是暂存单元，不是哨兵，当j<=0时，插入位置已到
    for(d= n/2; d>=1; d=d/2)     //步长变化
        for(i=d+1; i<=n; ++i)
            if(A[i]<A[i-d]){     //需将A[i]插入有序增量子表
                A[0]=A[i];                  //暂存在A[0]
                for(j= i-d; j>0 && A[0]<A[j]; j-=d)
                    A[j+d]=A[j];            //记录后移，查找插入的位置
                A[j+d]=A[0];                //插入
            }//if
}
```

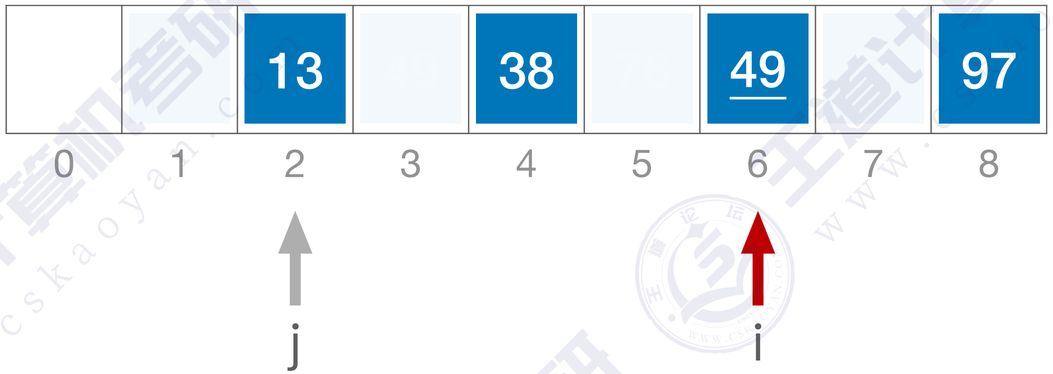
第二趟：d=2



算法实现

```
//希尔排序
void ShellSort(int A[],int n){
    int d, i, j;
    //A[0]只是暂存单元，不是哨兵，当j<=0时，插入位置已到
    for(d= n/2; d>=1; d=d/2)     //步长变化
        for(i=d+1; i<=n; ++i)
            if(A[i]<A[i-d]){     //需将A[i]插入有序增量子表
                A[0]=A[i];                  //暂存在A[0]
                for(j= i-d; j>0 && A[0]<A[j]; j-=d)
                    A[j+d]=A[j];            //记录后移，查找插入的位置
                A[j+d]=A[0];                //插入
            }//if
}
```

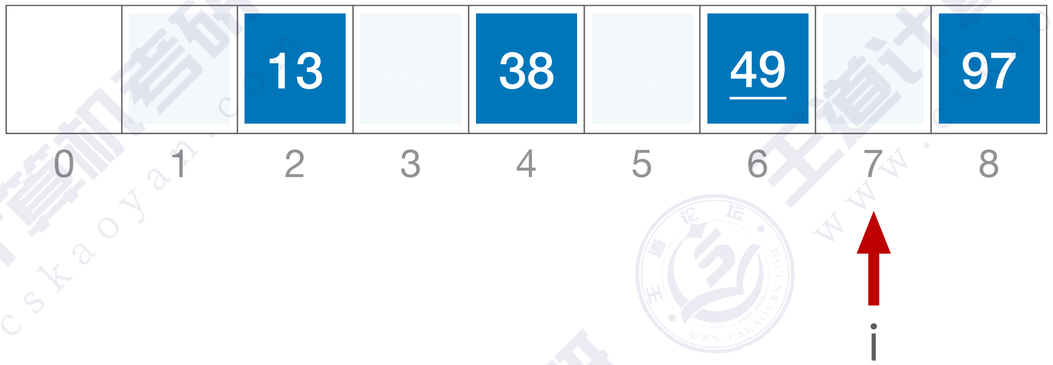
第二趟：d=2



算法实现

```
//希尔排序
void ShellSort(int A[],int n){
    int d, i, j;
    //A[0]只是暂存单元，不是哨兵，当j<=0时，插入位置已到
    for(d= n/2; d>=1; d=d/2)     //步长变化
        for(i=d+1; i<=n; ++i)
            if(A[i]<A[i-d]){     //需将A[i]插入有序增量子表
                A[0]=A[i];                  //暂存在A[0]
                for(j= i-d; j>0 && A[0]<A[j]; j-=d)
                    A[j+d]=A[j];            //记录后移，查找插入的位置
                A[j+d]=A[0];                //插入
            }//if
}
```

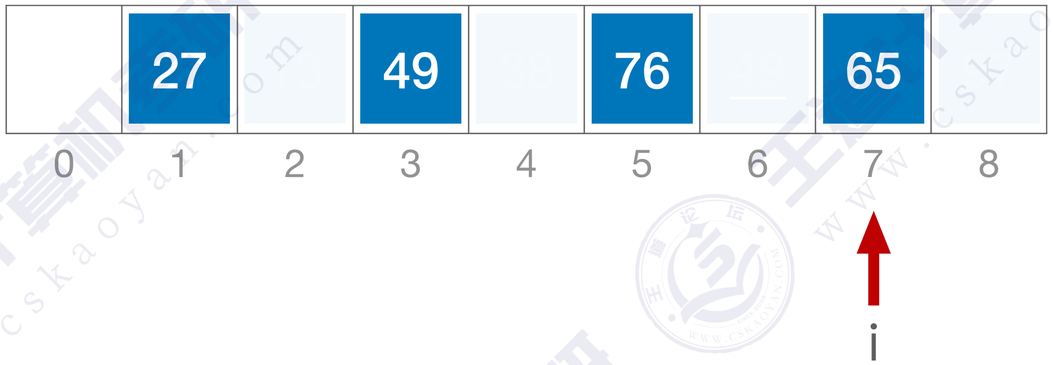
第二趟：d=2



算法实现

```
//希尔排序
void ShellSort(int A[],int n){
    int d, i, j;
    //A[0]只是暂存单元，不是哨兵，当j<=0时，插入位置已到
    for(d= n/2; d>=1; d=d/2)     //步长变化
        for(i=d+1; i<=n; ++i)
            if(A[i]<A[i-d]){     //需将A[i]插入有序增量子表
                A[0]=A[i];                  //暂存在A[0]
                for(j= i-d; j>0 && A[0]<A[j]; j-=d)
                    A[j+d]=A[j];            //记录后移，查找插入的位置
                A[j+d]=A[0];                //插入
            }//if
}
```

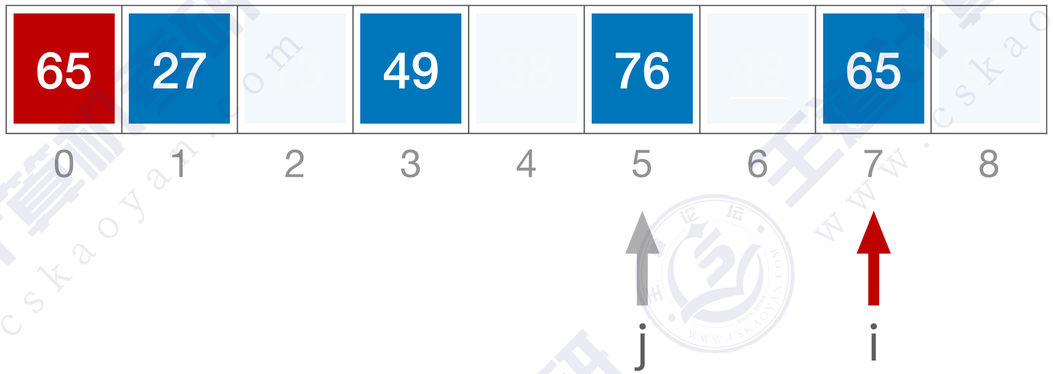
第二趟：d=2



算法实现

```
//希尔排序
void ShellSort(int A[],int n){
    int d, i, j;
    //A[0]只是暂存单元，不是哨兵，当j<=0时，插入位置已到
    for(d= n/2; d>=1; d=d/2)     //步长变化
        for(i=d+1; i<=n; ++i)
            if(A[i]<A[i-d]){     //需将A[i]插入有序增量子表
                A[0]=A[i];                  //暂存在A[0]
                for(j= i-d; j>0 && A[0]<A[j]; j-=d)
                    A[j+d]=A[j];            //记录后移，查找插入的位置
                A[j+d]=A[0];                //插入
            }//if
}
```

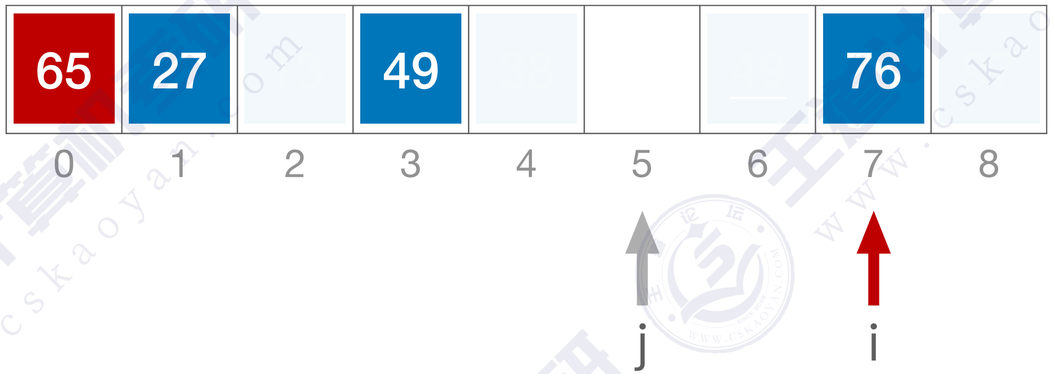
第二趟：d=2



算法实现

```
//希尔排序
void ShellSort(int A[],int n){
    int d, i, j;
    //A[0]只是暂存单元，不是哨兵，当j<=0时，插入位置已到
    for(d= n/2; d>=1; d=d/2)     //步长变化
        for(i=d+1; i<=n; ++i)
            if(A[i]<A[i-d]){     //需将A[i]插入有序增量子表
                A[0]=A[i];                  //暂存在A[0]
                for(j= i-d; j>0 && A[0]<A[j]; j-=d)
                    A[j+d]=A[j];            //记录后移，查找插入的位置
                A[j+d]=A[0];                //插入
            }//if
}
```

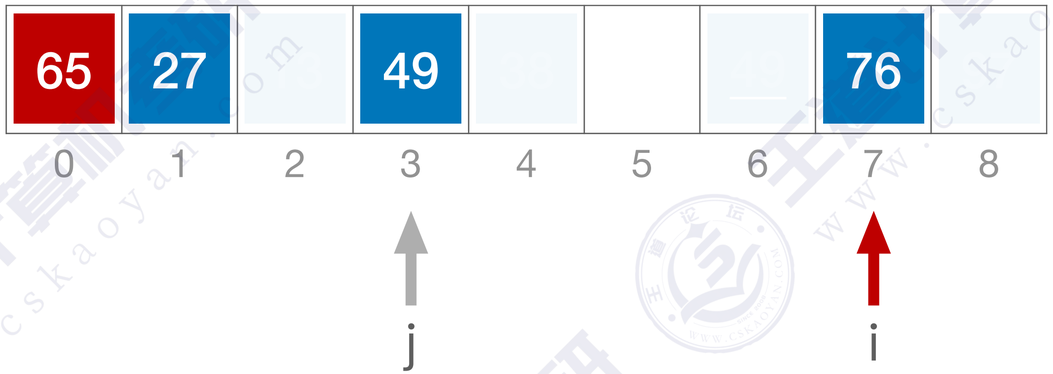
第二趟：d=2



算法实现

```
//希尔排序
void ShellSort(int A[],int n){
    int d, i, j;
    //A[0]只是暂存单元，不是哨兵，当j<=0时，插入位置已到
    for(d= n/2; d>=1; d=d/2)     //步长变化
        for(i=d+1; i<=n; ++i)
            if(A[i]<A[i-d]){     //需将A[i]插入有序增量子表
                A[0]=A[i];                  //暂存在A[0]
                for(j= i-d; j>0 && A[0]<A[j]; j-=d)
                    A[j+d]=A[j];            //记录后移，查找插入的位置
                A[j+d]=A[0];                //插入
            }//if
}
```

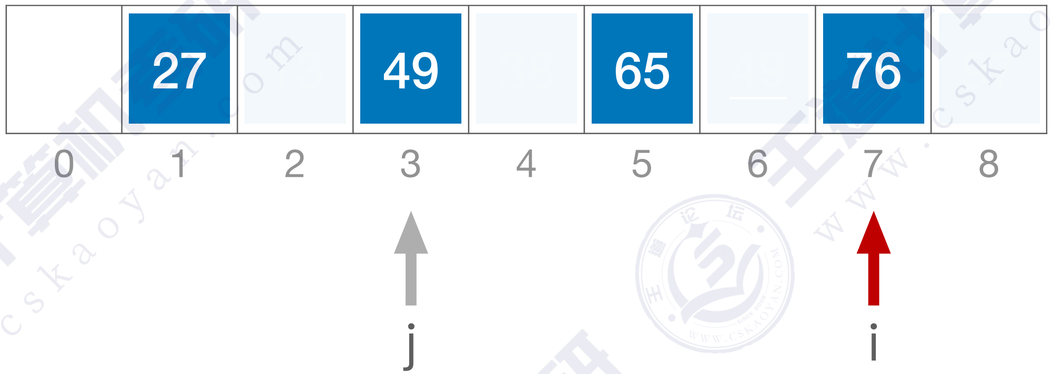
第二趟：d=2



算法实现

```
//希尔排序
void ShellSort(int A[],int n){
    int d, i, j;
    //A[0]只是暂存单元，不是哨兵，当j<=0时，插入位置已到
    for(d= n/2; d>=1; d=d/2)     //步长变化
        for(i=d+1; i<=n; ++i)
            if(A[i]<A[i-d]){     //需将A[i]插入有序增量子表
                A[0]=A[i];                  //暂存在A[0]
                for(j= i-d; j>0 && A[0]<A[j]; j-=d)
                    A[j+d]=A[j];            //记录后移，查找插入的位置
                A[j+d]=A[0];                //插入
            }//if
}
```

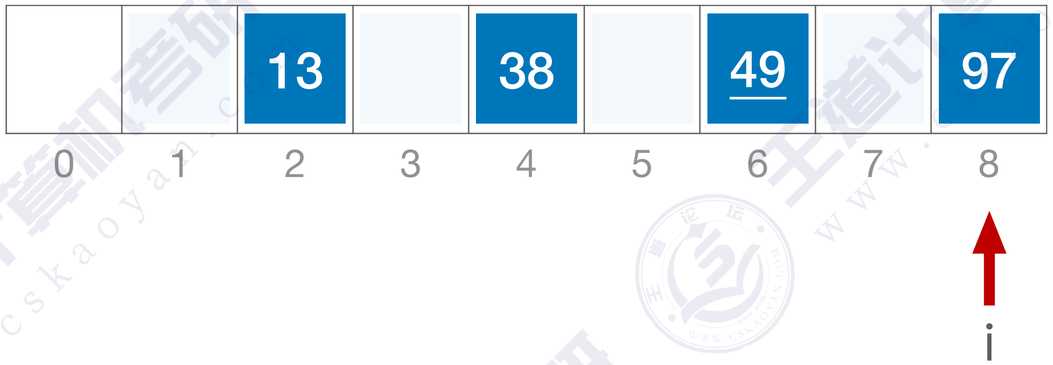
第二趟：d=2



算法实现

```
//希尔排序
void ShellSort(int A[],int n){
    int d, i, j;
    //A[0]只是暂存单元，不是哨兵，当j<=0时，插入位置已到
    for(d= n/2; d>=1; d=d/2)     //步长变化
        for(i=d+1; i<=n; ++i)
            if(A[i]<A[i-d]){     //需将A[i]插入有序增量子表
                A[0]=A[i];                  //暂存在A[0]
                for(j= i-d; j>0 && A[0]<A[j]; j-=d)
                    A[j+d]=A[j];            //记录后移，查找插入的位置
                A[j+d]=A[0];                //插入
            }//if
}
```

第二趟：d=2



算法实现

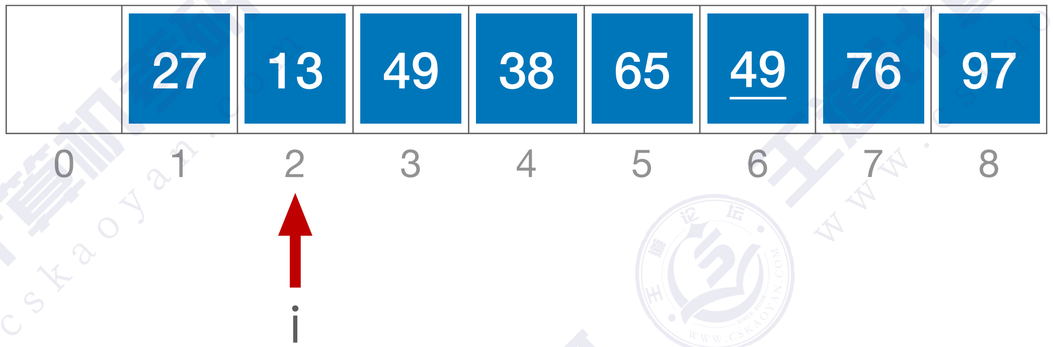
```
//希尔排序
void ShellSort(int A[],int n){
    int d, i, j;
    //A[0]只是暂存单元，不是哨兵，当j<=0时，插入位置已到
    for(d= n/2; d>=1; d=d/2)     //步长变化
        for(i=d+1; i<=n; ++i)
            if(A[i]<A[i-d]){     //需将A[i]插入有序增量子表
                A[0]=A[i];                  //暂存在A[0]
                for(j= i-d; j>0 && A[0]<A[j]; j-=d)
                    A[j+d]=A[j];            //记录后移，查找插入的位置
                A[j+d]=A[0];                //插入
            }//if
}
```

第二趟：d=2

算法实现

```
//希尔排序
void ShellSort(int A[],int n){
    int d, i, j;
    //A[0]只是暂存单元，不是哨兵，当j<=0时，插入位置已到
    for(d= n/2; d>=1; d=d/2)     //步长变化
        for(i=d+1; i<=n; ++i)
            if(A[i]<A[i-d]){     //需将A[i]插入有序增量子表
                A[0]=A[i];                  //暂存在A[0]
                for(j= i-d; j>0 && A[0]<A[j]; j-=d)
                    A[j+d]=A[j];            //记录后移，查找插入的位置
                A[j+d]=A[0];                //插入
            }//if
}
```

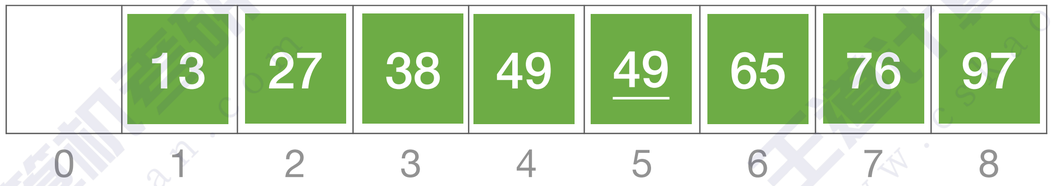
第三趟：d=1



算法实现

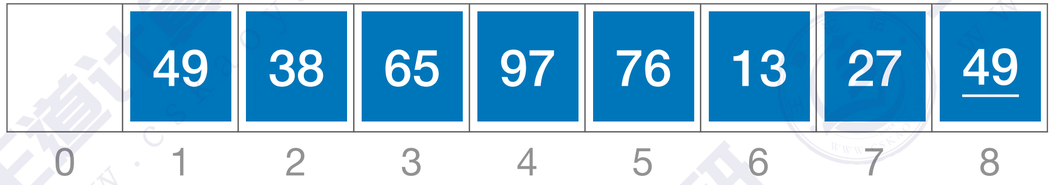
```
//希尔排序
void ShellSort(int A[],int n){
    int d, i, j;
    //A[0]只是暂存单元，不是哨兵，当j<=0时，插入位置已到
    for(d= n/2; d>=1; d=d/2)     //步长变化
        for(i=d+1; i<=n; ++i)
            if(A[i]<A[i-d]){     //需将A[i]插入有序增量子表
                A[0]=A[i];                  //暂存在A[0]
                for(j= i-d; j>0 && A[0]<A[j]; j-=d)
                    A[j+d]=A[j];            //记录后移，查找插入的位置
                A[j+d]=A[0];                //插入
            }//if
}
```

第三趟：d=1



算法性能分析

空间复杂度：O(1)



第一趟：$d_1=n/2=4$

第二趟：$d_2=d_1/2=2$

第三趟：$d_3=d_2/2=1$

第一趟：$d_1=3$

第二趟：$d_2=1$

时间复杂度：和增量序列 $d_1, d_2, d_3\ldots$ 的选择有关，目前无法用数学手段证明确切的时间复杂度

最坏时间复杂度为 $O(n^2)$，当n在某个范围内时，可达$O(n^{1.3})$

算法性能分析

原始序列：

第一趟：d=2

第二趟：d=1

稳定性：不稳定！

适用性：仅适用于顺序表，不适用于链表

知识回顾与重要考点